INTERNATIONAL AIR TRANSPORT ASSOCIATION

Common Use Self Service (CUSS) Technical Specification

Revision: 1.3

Date: June 2013

Revision Table

Revision	Date	Change
0.5	2003-04-25	Initial Release
1.0	2003-05-16	Updated the Preface; added Index; reformatted document.
1.2	2009-03-23	CUSS 1.2 submitted to IATA CUSSMG (now PEMG)
1.2	2009-05-25	Ratified by IATA for publication
1.3	2013-05-01	CUSS 1.3 Technical Specification finalized
1.3	2013-06-17	CUSS 1.3 Technical Specification published

Table of Contents

List of Figures

Introduction and Preface

About This Document

This document describes the IATA CUSS Technical specifications, a standard that allows multiple airlines to share one physical kiosk to offer self-services to their passengers. These services include, but not limited to, check-in functionality. The standard also allows airlines to develop CUSS-compliant applications that are able to run on any kiosk whose platform is CUSS-compliant.

This specification has been approved by the IATA Passenger Experience Management Group (PEMG) based on the recommendation of the Common Use Working Group (CUWG) and the CUSS Technical Solution Group (TSG-CUSS.)

Between releases of the CUSS Technical Specification, corrections and clarifications to this technical specification are published in a separate errata document. The current version of that errata document is available from the IATA PEMG Extranet (as described elsewhere in this document:

IATA_CommonUseSelfService_TechnicalSpec_CUSS_1.3_errata.pdf
CUSS Technical Specification 1.3: Errata and Technologies Updates

Versions of the CUSS Technical Standard

This version of the document defines the CUSS Technical Specification version 1.3, and replaces all previous versions of this document. It is published in June 2013. As of version 1.3, the CUSS Technical Specification introduces a version lifecycle policy that deprecates previous versions of the specification:

1. *At any given time, the current TS version and two prior versions remain active*
2. *When a new TS version is published, the oldest active version remains active for one year*

For example, the current version of CUSS-TS is CUSS 1.2. That means that CUSS 1.0, CUSS 1.1 and CUSS 1.2 are valid revisions of the TS. If CUSS 1.3 is published in June 2013, then CUSS 1.0 will become invalid in June 2014.

Effective 01 June 2014, the CUSS Technical Specification version 1.0 is deprecated. Sites that do not comply with version 1.1 or later will no longer be IATA RP1706c CUSS compliant as of that date.

Version 1.1 will be deprecated one year after the future publication of CUSS Technical Standard 1.4 (when and if that version is released.)

About IATA

International air transport is one of the most dynamic and fastest-changing industries in the world. It needs a responsive, forward-looking and universal trade association, operating at the highest professional standards. IATA is that association.

Originally founded in 1919, IATA brings together approximately 280 airlines, including the world's largest. Flights by these airlines comprise more than 95 percent of all international scheduled air traffic.

Since these airlines face a rapidly changing world, they must cooperate in order to offer a seamless service of the highest possible standard to passengers and cargo shippers. Much of that cooperation is expressed through IATA, whose mission is to "represent and serve the airline industry".

About PEMG and CUWG

IATA's Passenger Experience program addresses the end to end passenger journey from ticket purchase through to arrival at destination. It comprises a range of projects to improve the travel experience and help reduce unnecessary operational costs to the industry. One of the primary delivery channels is self-service options for passengers where it makes sense. In process areas controlled by government authorities, such as Security, Immigration and Customs, Passenger Experience will improve the facilitation of these processes by harmonizing passenger data requirements and enhancing passenger preparedness to reduce queues and process times.

The main functions of the Management Group are:

- Set direction and policy for all areas within Passenger Experience
- Orovide oversight and governance for the constituent working groups
- Review and approve proposed additions, changes and deletions to Standards within PEMG and the constituent working groups as well as any future products and/or PEMG activities
- Submit an annual report of its activities to the JPSC meeting
- Liaise closely with other ATA and IATA Committees impacting on PEMG Standards

Membership of PEMG is open to IATA & ATA Members, IATA Strategic Partners and members of Airports Council International (ACI). In addition, membership of the Passenger

Facilitation Working Group (PFWG) is open to government agencies. To sign up for access to the IATA PEMG Extranet site, submit your information via this URL:

https://www2.iata.org/registration/Getemailpage.aspx?siteUrl=PEMG

The Common Use Working Group (CUWG) is part of the IATA Passenger Experience Management Group. The main functions of the working group are:

- *To review and approve proposed additions, changes and deletions to Common Use standards including RP 1797 & RP 1706 as well as any future standards relating to products used in the airport common use environment.*
- *To submit an annual report of its activities to the Passenger Services Conference (PSC).*
- *To liaise closely with other bodies, including the Air Transport Association (ATA), Airports Council International (ACI) and IATA Committees impacting on Common Use Standards.*

Recommended practices, technical specifications and certification criteria developed by this group are published in the IATA CUSS Manual (this document) and the IATA CUPPS Manual – the publications are available on the IATA PEMG Extranet.

https://extranet2.iata.org/sites/pemg/common-use-wg/default.aspx

Intended Audience

This document is intended to be used by software designers and developers who want to produce platforms and applications that comply with CUSS standard. Application developers who are already familiar with the concepts of CUSS may wish to jump directly to Chapters 7 and 8 (new in CUSS 1.2 and new in CUSS 1.3) for new information about how to find and use real devices on a CUSS platform.

While this Technical Specification is a document for software practitioners, IATA has also published a separate Common Use Self-Service implementation guide that provides more broad information on deploying CUSS at the airport. This document is available at

http://www.iata.org/whatwedo/stb/cuss/Pages/index.aspx

Organization of This Document

This document is divided in the following chapters and appendices:

Chapter 1 – Architecture Overview, describes the overall CUSS kiosk architecture, including a general description of a CUSS platform and a CUSS application.

Chapter 2 – Interface Overview, gives a general overview of the interfaces between a CUSS application and a CUSS platform.

Chapter 3 – Interface Definition, defines all interfaces used in CUSS, mainly the Application Manger Interface, the System Manager Interface, the Device Components Interface, and the Event Listener Interface used by the CUSS platform to communicate with CUSS applications (kiosk applications and system manager applications).

Chapter 4 – Real Component Characteristics, lists all CUSS real components (mandatory, recommended, and optional) as well as their hardware and software characteristics.

Chapter 5 – Virtual Component Characteristics, lists all the characteristics of CUSS virtual components used to represent the CUSS real components.

Chapter 6 -- Extended Device & Media Type Handling, indicates how new and extended types of data are handled exchanges by applications and the platform.

Chapter 7 -- Real Device Programming Guide, explains how a CUSS application developer should interact with common devices in a CUSS environment.

Chapter 8 -- The CUSS FOID Addendum, explains how a CUSS platforms and applications must request and process sensitive payment card track data.

Chapter 9 -- Application Automatic Remote Updates, indicates the methods and operational guidelines and application must follow to automatically update itself.

Appendix A – Return, Event, and Status Codes, lists all function return codes, event codes and status codes used in the CUSS standard.

Appendix B – Component Mappings, includes a table that maps all CUSS real components into their corresponding CUSS virtual components.

Appendix C – IDL Listings, contains all the CUSS IDL files and schema definitions that comprise the CUSS standard interfaces.

Appendix D – AEA Standard Profiles, refers to the AEA standards used in CUSS and provides many clarifications to parts of the AEA specification that are ambiguous.

Appendix E – Presentation Technologies and Standards, lists all the presentation service technologies that should be supported by CUSS 1.2 platforms.

Appendix F – Self-Certification Criteria, refers to a separate document that includes all the certification criteria to be satisfied as part of certification of CUSS platforms and applications.

Appendix G – Printer Stock and Document Types, explains the different types of documents available on a CUSS kiosk.

Appendix H – Extended Data Type List, enumerates the latest extended DS_TYPES described in Chapter 6.

Appendix I – Application Updates and Distribution, provides guidelines about how applications are packaged and updated..

Appendix J – Upgrading to a new version of CUSS, which discusses the platform and application changes need to adapt to CUSS 1.3.

Appendix K – The CUSS Technical Specification FIles, which provides a concise list of the files that comprise the CUSS Technical Specification, and their purpose..

Glossary, includes a glossary for acronyms used through out the document.

Associated Documents

The CUSS technical specifications document is part of the IATA CUSS Manual, which also includes the following publications by the CUSS Management Group:

- CUSS Interface Defintion Language (IDL) files
- CUSS XML Message schema (XSD) files
- CUSS Certification Document
- CUSS Self-Certification Criteria
- Service Level Agreement Templates

References

The following standard and publications have been referred to in this document:

- AEA – ATB Technical specs- Amended August 2002, 2008, 2009
- AEA – Parametric Baggage Tag Data Concept – August 2002, 2008, 2009
- AEA - Self Service Specifications - August 2001
- AEA2012-2 – Self Bag Drop – March 2013
- EMV – Visa Integrated Circuit Card standard version 1.3.2
  - Application Overview
  - Card (ICC) Specification
  - Terminal Specification
- HTML Specification, version 4.0 or later, by W3C
- IATA Recommended Practices 1706c, 1706d, 1706e, 1720a, 1723
- IATA Resolutions 722c, 722d, 722e
- IATA Resolution 792 (BCBP)
- ISO 7810-7811, 7812, 7816
- ISO 8859-1 Latin 1
- ISO/IEC 10646-1 second edition
- Java™ 2 Platform, Standard Edition, version 1.3.1, by Sun Microsystem
- Java™ 2 Platform, Standard Edition, version 1.5.0, by Sun Microsystems
- Java™ 2 Platform, Standard Edition, version 7, by Oracle

- Scalable Vector Graphics (SVG), version 1.1 or later, by W3C
- CORBA specifications, version 2.3 or later, by OMG
- Unicode Standard version 3.0.0, by Unicode, Inc.

Participation in the CUSS Technical Standard

The IATA Passenger Experience Management Group maintains an extranet for Common Use Working Group activities. This extranet portal provides for ongoing discussion of issues relating to the CUSS Technical Standard and future versions. To access the PEMG website, register at:

https://www2.iata.org/registration/Getemailpage.aspx?siteUrl=PEMG

IATA Members and Partners should contact their IATA representative to understand how they can become involved with the Passenger Experience Management Group and associated subcommittees, including the CUSS Technical Solution Group.

Airport Common Use: CUSS and CUPPS

IATA current has defined two standards for Common Use airports. The first is Common Use Self Service (CUSS) which is designed for self-service kiosk operations and creating applications used directly by airline customers. CUSS has been in production since 2003 and is an IATA standard. This document is its Technical Specification.

The second is Common Use Passenger Processing Systems (CUPPS) which is primarily intended to support applications created for use by air travel staff at check-in counters, boarding gates, and other travel areas. CUPPS has been in production since 2009 and is a standard endorsed by IATA, ATA, and ACI. CUPPS has its own Technical Specification documents and processes.

The CUPPS standard is the outcome of a formal set of business requirements, crafted specifically with “behind the counter” air travel staff usage in mind, not “customer facing” self-service usage. For this reason, the resulting CUPPS standard does not yet explicitly enumerate or meet any requirements that exist only in kiosk environments.

For this reason, the content of the CUPPS standard does not address all the requirements of the large installed based of CUSS kiosks and applications, which have been deployed since 2003.

IATA's goal is to eventually merge the CUSS and CUPPS Technical Specifications, and the supporting Compliance and Certification processes into a single, unified Common Use Standard.

The CUSS and CUPPS standards overlap in some areas, and are distinct in others. Together, they define consistent and predictable environments on which airlines can deploy applications into the airport environment.

No timeline has yet been set for the integration of CUSS and CUPPS, and as of publication of this document CUSS 1.3 remains the Common Use standard for self-service devices at the airport.

For more information on CUPPS, please see the IATA PEMG Extranet site.

About CUSS 1.3 and Document Changes

The CUSS Technical Solution Group (TSG-CUSS) received mandate at IATA PEMG05/SEA to update the CUSS Technical Standard (CUSS-TS) from version 1.2 to version 1.3 with the following goals[1]:

1) Merge the CUSS FOID Addendum into the CUSS Technical Specification (Chapter 8)
2) Define a new unified Baggage Scale/Conveyor interface with both a CUSS Component Mode interface, as well as an AEA-SBD passthrough interface (Chapter 7)
3) Define a new Payment Device interface (Chapter 7)
4) Define capabilities for Automatic Remote Update (ARU) for applications (Chapter 9)
5) Update the Technologies List to current versions (Appendix E), in particular the major updates to:
    i. Java Runtime Environment 7
    ii. Internet Explorer 8 as the default "Standard CUSS Browser "
    iii. Adobe Flash 11.7.7
    iv. Silverlight 5
6) Upgrade printing support to AEA2009 and enable additional AEA commands

[1] Refer to the PEMG05-CUSS-TSG technical minutes and TSG Status Update to CUWG

This updated Technical Specification document fulfills those goals and defines version 1.3 of the IATA Common Use Self-Service (CUSS) standard. This single CUSS-TS document presents the proper and mandatory behaviour of CUSS 1.3 compliant platforms and applications.

The following changes have been made in updating this specification document from CUSS 1.2 to CUSS 1.3 (see the next section for changes from CUSS 1.0 to CUSS 1.2):

- Update revision number, date, and year of IATA©, and change CUSSMG references to PEMG/CUWG
- New "About CUSS 1.3 and Document Changes" introduction
- Add the Version Lifecycle statement about the deprecation of CUSS 1.0.
- Include basic information on CUPPS in the introduction
- Updated Section 1.7 to include more information on application and platform responsibilities for securing sensitive payment card data.
- Add RI, RC, PV, EP and ES to supported AEA commands
- New Section 2.4.4 (Modes of Operation for Applications) and Section 2.4.5 (Special State Transitions and Notification Strings) from the CUSS 1.0 Addendum
- In Chapter 7, added the revised Integrated Baggage Conveyor definition, based on AEA2012-2 for Self Bag Drop (SBD) devices.
- In Chapter 7, added a new dedicated Baggage Scale definition for weight scales not connected to baggage systems.
- Add the new Payment Device interface to Chapter 7 including an XML schema for transaction data communication.
- Merge the CUSS FOID Addendum into the document as a new Chapter 8, and include specific updates as included in v1.3 of the CUSS FOID Addendum.
- Add Chapter 9 defining how application Automatic Remote Update should take place in CUSS 1.3
- Update Appendix E to list the current tools, technology and runtime environment available to CUSS applications at the kiosk
- Update Appendix I to reflect changes to guidelines about application updates and distribution in the context of Automatic Remote Update (ARU)
- Update the interface definition files comps.idl, types.idl, codes.idl and characteristics.idl to support changes in the Technical Specification
- Add the new CUSS.PAYMENT.XSD file defining the data format for the Payment Interface

- Add the new CUSS.SBD. XSD file defining the data format for the data messages for Baggage Conveyors and Scales, including weight, alibi, dimension, RFID, and other information.
- Add the new ACTIVE_UNAVAILABLE and ACTIVE_ACTIVE transitions.
- Clarification of DS_CORRUPTED behavior when returning data to the application
- Define how barcode scanners can return multiple tracks of data in response to reader devices that detect and scan multiple barcodes on a document.
- The existing Conveyor component and definition that existed in CUSS 1.2 remains in the CUSS 1.3 IDLs but has been deprecated. This component is replaced by new Integrated Baggage Conveyor and Baggage Scale component definitions.

For a quick roadmap for upgrading kiosks and platforms to CUSS 1.3, please read *Appendix J: upgrading to CUSS* 1.3.

About CUSS 1.2 and Document Changes

The CUSS Technical Solution Group (TSG-CUSS) received mandate at IATA CUSSMG24 to update the CUSS Technical Standard (CUSS-TS) from version 1.0 to version 1.2 with the following goals[2]:

1) Update the Technologies List to current versions (Appendix E)
2) Merge Addendum document into Technical Specification (see Appendix J)
3) Include Real Device Behaviour Clarification document (Chapter 7)
4) New section covering Change Control documents from CUSSMG23 (Appendix I)
5) Include PDF printing support as a printing option for GPP printers (Chapter 6.4)
6) Add data security requirements for PCI DSS (Section 1.7)
7) Clearly state the upgrade requirements and compatibility of 1.2 vs. 1.0/1.1 (Appendix J)
8) Add AEA2008 printer support and a 2D barcode scanner as **mandatory** equipment for CUSS-1.2 kiosks, to support IATA Resolution 792.

This updated Technical Specification document fulfills those goals and defines version 1.2 of the IATA Common Use Self-Service (CUSS) standard. It combines previous CUSS-TS documents into an updated single reference (this document) which presents the proper and mandatory behaviour of CUSS 1.2 platforms and applications:

- Common Use Self Service (CUSS) Technical Specification Revision 1.3
- CUSS 1.0 Addendum Document
- CUSS 1.0 Clarification of IATA CUSS Real Device to Virtual Component Mapping

Within this specification, there are references to CUSS 1.0 Addendum A.1.??. These indications are only to provide a source that justifies the change to the specification for CUSS 1.2. Readers do NOT need to refer back to the CUSS 1.1 Addendum document for further information.

[2] Refer to the CUSSMG24-TSG technical minutes and TSG Status Update to CUSSMG

Please note: This CUSS-TS 1.2 document is based on a recovered version (from PDF) of the original CUSS 1.0 specification. Some formatting in the original CUSS 1.0 document may have been lost in this conversion process.

The following changes have been made in updating this specification document from CUSS 1.0 to CUSS 1.2:

- Update revision number, date, and year of IATA©
- New About CUSS 1.2 Introduction
- New Section 1.7 providing a brief discussion on data security
- Add ZS and AV to supported AEA commands.
- New Sections 2.1.1 and 2.1.2 with more information on CORBA TCP/IP requirements
- New section 2.4.3.10 explaining the Periodic/Automatic restart for CUSS applications
- New Section 2.4.4 (Modes of Operation for Applications) and Section 2.4.5 (Special State Transitions and Notification Strings) from the CUSS 1.0 Addendum
- New Section 3.2.3 with information on how to handle components whose behavior can depend on other components, and new Section 3.6.11 explaining how to use printers which have a limited capacity to stack or hold documents.
- New Section 3.8 showing how MediaInput devices (such as card readers) generate MEDIA events.
- New Chapter 6 (Extended Device & Media Type Handling) and Appendix H (Extended Data Types List) defining the use of extended data types in CUSS 1.2
- New Section 6.4 permitting PDF printing if kiosks are equipped with a GPP
- New Chapter 7 (Real Device Programming Guide) explaining how CUSS application developers should interact with kiosk devices.
- Updates to Appendix B to indicate how kiosk devices are used in CUSS.
- New Appendix D content to include all AEA clarifications from the CUSS 1.0 Addendum.
- Expanded Appendix E explaining the tools, technology and runtime environment available to CUSS applications at the kiosk.
- New Appendix G relating to physical document characteristics from the CUSS 1.0 Addendum.
- New Appendix I providing guidelines about application updates and distribution.
- New Appendix J cross-referencing CUSS 1.0/1.1 Addendum entries with CUSS 1.2 specification changes.

- Updates to level(), enable(), disable(), offer() and receive() removing some data and even handling ambiguity and providing some real device examples.
- Remove the Index section (its formatting was lost in the conversion from PDF.
- Add Baggage Conveyor components to comps.idl and characteristics.idl, and new constants (DS_TYPES, ACTIVE_ACTIVE, etc) to codes.idl
- Moved Barcode Scanner into the Mandatory components Section 4.1. In CUSS 1.0 and 1.1, a barcode scanner device is optional. For CUSS 1.2 compliance, a kiosk must include a 2D barcode scanner.

Chapter 7 is a significant addition to the CUSS Technical Specification. It is written to provide more practical information to application developers on how to find, set up, and use different types of devices on a CUSS kiosk. This new chapter combines information from earlier Chapters 1-5 and presents a clearer picture of how to use CUSS devices.

For a quick roadmap for upgrading kiosks and platforms to CUSS 1.2, please read *Appendix J: upgrading to new version of CUSS* in Version 1.2 of the CUSS Technical Specification.

Ch 1: Architecture Overview

This section describes the overall architecture of the Common Use Self Service (CUSS) standard. Currently, this standard only covers Self Service kiosks. Other Self Service facilities will be incorporated into the CUSS standard as soon as they become available within the travel industry. This section consists of the following subsections:

CUSS Principles

CUSS Kiosk Architecture

CUSS Platform Hardware

CUSS Platform Software

Kiosk Application (AL application)

Complete CUSS Environment

1.1 CUSS Principles

The CUSS standard was developed according to the following principles:

The standard is operating system independent

The platform providers are not forced to use a particular operating system; they have the freedom to use the operating system that is most effective for them, and for the application providers they would like to serve.

No specific hardware platform is assumed

The CUSS standard doesn't state any particular processor architecture. In addition, the hardware of a standard compliant kiosk system should not be limited to particular devices (i.e. devices from specific vendors). Platform providers are allowed to use the hardware devices they choose as long as these devices provide the functionality that is required to support the device components interface.

The standard allows for vendor independence

In addition to the hardware and software, no specific vendor of kiosks is mandated by the CUSS standard. Any vendor providing a competitive product may be used.

Self Service applications are platform independent

CUSS certified applications should be capable of being run on any CUSS certified platform. A platform must support multiple concurrent applications. Refer to the ‘CUSS Certification’ document for more details.

1.2 CUSS Kiosk Architecture

A CUSS kiosk is intended for use in a self-service environment. In a typical self-service process, the customer gives some information and the kiosk returns information or hands out documents. Therefore the kiosk is equipped with reading devices and printing devices to make out the documents. The rest is a normal PC or NC, equipped with at least a touch screen for user interaction.
Platform providers offer the CUSS kiosk, and application providers share it to run their kiosk applications on.
A kiosk is composed of a platform and one or more applications. To realize a common kiosk platform, the hardware device access layer has to be hidden away from the applications that run on such a kiosk. A hardware abstraction layer is introduced that offers common interfaces for the different hardware devices within a kiosk to the application. The result is the following three-layered architecture (Figure 1).

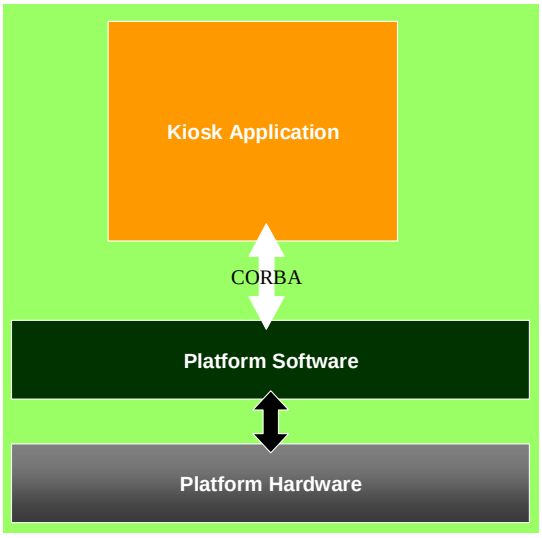


Figure 1 Three-Layered CUSS Kiosk Architecture

The lowest layer consists of the kiosk platform hardware. The next layer is the platform software, consisting of the platform environment in general (operating system, software plug-ins, etc.), the Application Manager, the system manager interface, the Common Launch Application, and the hardware abstraction layer containing the Device Components. Each device component drives a dedicated hardware device, and provides a standard interface to it. The third layer is one or more application(s) running on such a kiosk. The interface between a kiosk application and any platform component is based on CORBA and is defined in Section Ch 3:: Interface Definition.

1.3 CUSS Platform Hardware

CUSS Platform hardware consists of all hardware components included in the kiosk. This currently includes a normal PC/NC, a touch screen, a magnetic card reader and a boarding pass printer (AEA.) Other recommended and optional components could also be added to the kiosk. Please refer to Section Ch 4:: Real Component Characteristics, for more details.

1.4 CUSS Platform Software

The platform software consists of all the software included in the kiosk except those supplied by the application providers.

The platform software is responsible for managing the entire kiosk system including:

- instantiation and presentation of all platform processes including browser and device components,
- displaying common screens while no kiosk application is active
- providing data and statistical information to the remote management system via the system manager interface,
- controlling/monitoring components states, and
- managing system security.

These responsibilities are shared by various platform elements, namely, the platform environment, the application manager, the system manager interface, the common launch application, and the device components. The following sections describe these elements in more details.

1.4.1 Platform Software Environment

The platform provider is responsible of supplying the kiosk with the following software environment (see Figure 2): operating system, Internet browser, Java virtual machine, miscellaneous software containers and plug-ins (e.g. Macromedia shockwave player) as described in Appx E:: Technologies and Standards.

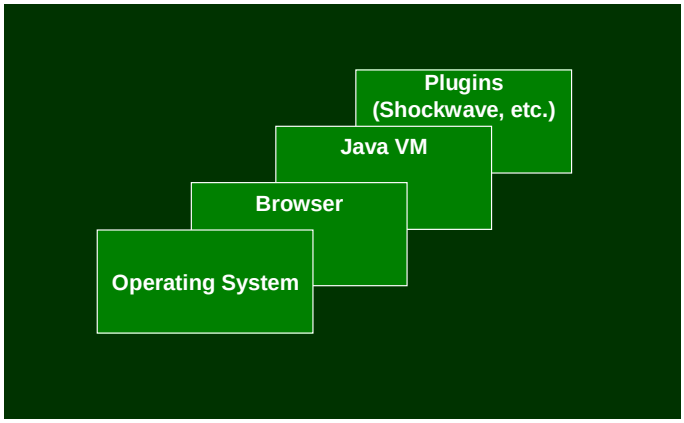


Figure 2 Platform Software Environment

1.4.2 CUSS Application Manager (CAM)

The CUSS Application Manager is responsible for controlling and scheduling the kiosk applications that are registered on a specific kiosk. This includes declaring individual or all applications stopped or suspended upon instruction from the system manager interface. For example, a platform provider may wish to stop all applications at night when the airport is not operational.
The application manager is also responsible for informing the Common Launch Application (refer to Section 1.4.4) to remove (or make un-selectable) application icons from the selection (launch) screen when individual applications are disabled, stopped, suspended or become unavailable for whatever reasons, or to display an appropriate general "kiosk not available" screen when all applications are disabled, stopped, suspended or unavailable.
As shown in Figure 3, the application manager is in charge of the event dispatcher of the public event channel, the platform software environment and component repository and the access control of the kiosk platform (including the device components).

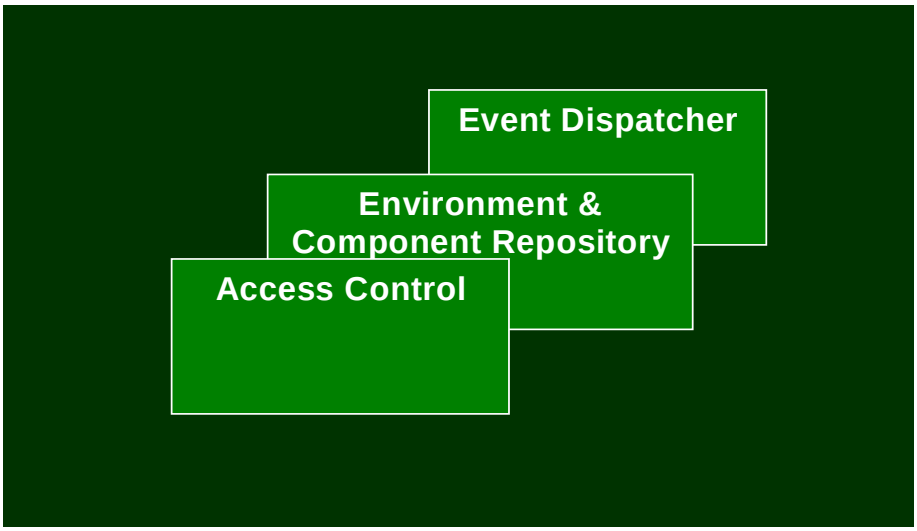


Figure 3 CUSS Application Manager

1.4.2.1 Event Dispatcher

All kiosk applications, whether they are in background or in foreground, may connect to the public event channel that will be serviced via the event dispatcher. In this way, background applications may inform the application manager that they are available (selectable on the launch screen) or unavailable, depending upon the devices they need, and the ones they can operate without. For example, assume the ATB reader reports its failure via the public event channel, then the application decides whether it is able to proceed without an ATB reader, if so, the application remains available, but if not, the application calls the application manager to make itself not available. In this example most applications may want to continue operating, since they can continue serving customers with electronic tickets (assuming that the card reader is still available).

1.4.2.2 Environment & Component Repository

The Environment & Component Repository is obtained from the application manager. It contains information about the platform environment itself (e.g. kiosk location, software environment, etc.) and a list of virtual components, along with their attributes, so that components can be selected by an application for operation. If a device component isn't registered in the component list, the appropriate hardware device is not present in the kiosk. It is then up to the application to decide whether the requested component is mandatory or optional for its proceeding. If it is optional, the application can still execute, but with limited functionality, depending on the devices that are present.

1.4.2.3 Access Control

The kiosk application has only restricted access to the underlying system, which is controlled by the platform (application manager and device components) to assure the security of the kiosk platform.
The application manager is also responsible for assuring that only the currently active kiosk application and its backend system(s) are able to access the device components of a kiosk. Therefore a mechanism is needed to ensure that only one kiosk application and its associated backend system can access the kiosk platform at a time. The application manager issues an application token whenever the kiosk application procedure initializes. While the application is active, this token is valid for full access of the device components. After the application has terminated (unloaded) or has re-initialized, the application manager invalidates the token.

1.4.3 System Manager Interface (SMI)

The system manager interface is a standard CORBA interface implemented on the kiosk, allowing for remote management of kiosk by providing the ability to control and monitor the platform to registered system managers via interfaces with the device components and the application manager functions in the platform. The responsibilities of the system manager interface includes:

- allowing a system manager to register its listener(s) for receiving general events, errors, alerts, alarms, etc.
- reporting errors, alerts, alarms encountered by device components (platform components)
- reporting normal events (e.g. Application-specific events to its system manger, application state changes to the SP system manager)
- gathering statistical information (platform provider's/service provider's scope)
- receiving commands (such as load/stop/suspend/resume) from service provider's system manager or application provider's system manager. These commands will be relayed to application manager to change the state of (a) particular application(s).

1.4.4 Common Launch Application (CLA)

The Common Launch Application, also known as Selection Interface Screen, is the application that is resumed during idle times, when no other kiosk application is active. It shows the common launch screen with all application providers' logos that have a kiosk application registered on the kiosk and that are currently available and selectable. The customer chooses the application provider's logo. This choice is reported to the application manager, which then activates the indicated kiosk application. In case of all the registered kiosk applications are unavailable, stopped, disabled or suspended, the common launch application will show "kiosk not available" type of screens. In general, the behavior of the common launch application changes when one or more kiosk applications change their states. Refer to Section 2.4.3 for descriptions of all application state transitions and their implications on the CLA behavior. The platform provider supplies the common launch application. Hence, the interface between CLA and other platform components (like CAM) is not specified in CUSS and left up to the platform provider.

1.4.5 Device Components

The kiosk application is not allowed to access the hardware devices directly. To realize this, the CUSS standard introduces a hardware abstraction layer that hides the proprietary device interfaces from the kiosk application. The kiosk application accesses the hardware devices through the device component interfaces of the platform (see Figure 4).

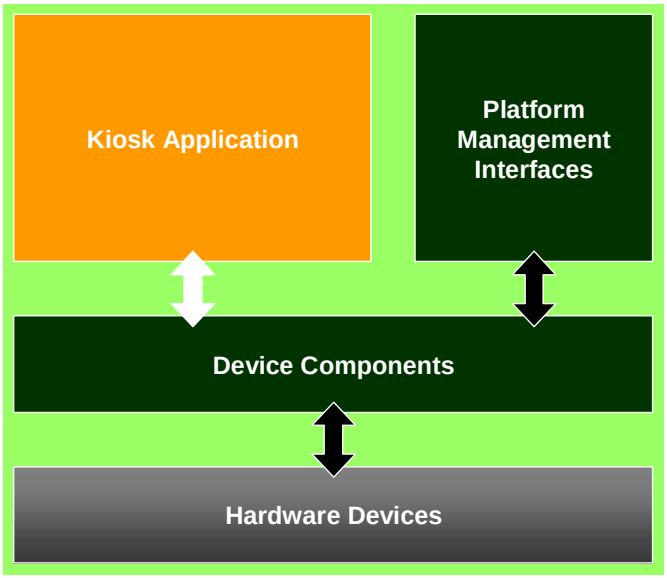


Figure 4 Device Components

A device component is an object that resides on the kiosk and is part of the platform software. It is independent from the kiosk application, and the interface is used to access the device. A device component can implement one or more interfaces, each of which is specific to a functionality that is offered by the associated hardware device. For instance, the device

component of an ATB reader/printer with an escrow for example provides at least three interfaces, one for the ATB reader functionality, one for the ATB printer functionality and one for the escrow.
A device component is not only an adapter to the associated hardware device; it is also responsible for controlling the state of the device. A kiosk application can either ask the device component for the state of the hardware device at the time it needs this information, or the application can assume that a device is working and gets its status if it is not. The application doesn't have to keep track of the device's state.
A device component is a distributed CORBA object; therefore, it is remotely accessible. This allows components of multi-tier applications to access the local component interfaces of the kiosk system. An access control mechanism (refer to Section 1.4.2.3) assures that only the backend of the currently active kiosk application has the permission to access the device components of the kiosk.

1.5 Kiosk Application (AL application)

The kiosk applications are the essential components of a CUSS kiosk system. They provide the functionality the kiosk offers to the customers using it. It depends on the kiosk application whether a kiosk can be used for airline check-in/ticketing or virtually any other services (e.g. ATM, TVM, etc.). There is no theoretical limitation about what and how many kiosk applications can be offered on a kiosk system. The kiosk applications are executed/controlled within an execution environment that is provided by the platform (See Figure 5).

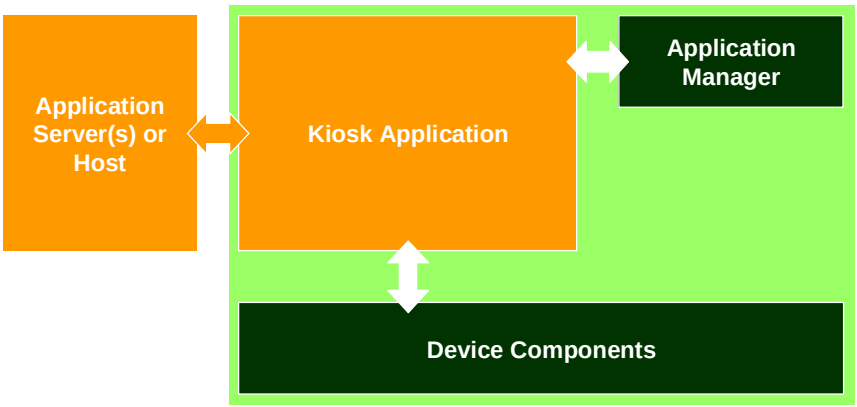


Figure 5 Kiosk Application and Associated Components

A kiosk application could be any range from a fat client, thin client to a very thin client. The thinner the client, the greater the portion of the application that resides and runs on an external back-end application-specific server. For more details on application architecture options, refer to Section 2.2: Application Architecture Options.

Note: Application clients running on a kiosk can be either Java-based or browser-based, where only cross-OS controls are allowed in the Standard CUSS Browser. Customized JVMs are not allowed and only one specific JVM version will be allowed in CUSS to

achieve compatibility among platforms and applications (Refer to Appx E: for more details on accepted JVM, browsers plug-ins, etc).

For performance reasons, all kiosk applications registered on a kiosk are started at a given time, usually power-up, and are run concurrently on that kiosk. Only one application is active at a time. Only the active kiosk application and its associated backend system are allowed to access the kiosk's device components, except for status listening.

1.6 Complete CUSS Environment

The previous sections introduced the components of the CUSS kiosk platform. Now we want to introduce the complete CUSS environment, the environment a CUSS compliant kiosk is part of. All elements of this environment have been introduced together with the platform components they interact with. This section puts these pieces together into a complete picture (Figure 6).

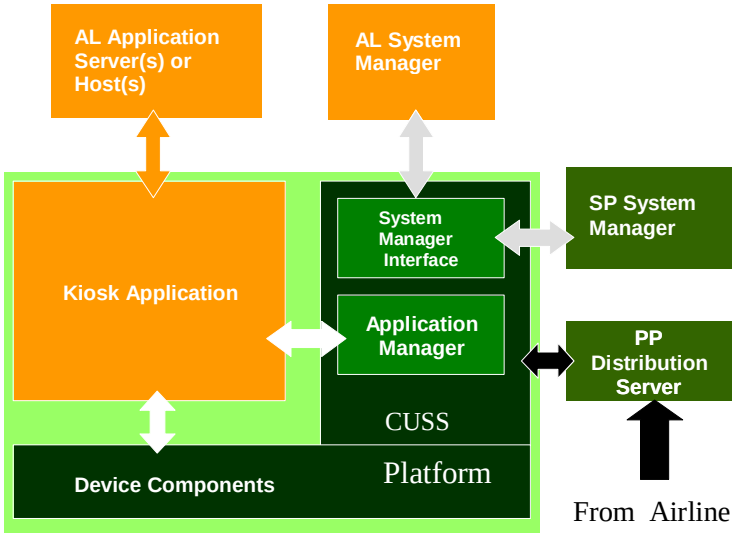


Figure 6 The Complete CUSS Environment

The CUSS environment is divided into two domains. One domain is the common environment, which is provided by a platform provider and managed by a service provider. This domain contains the dark colored (black and green) elements in the diagram (Figure 6). The other domain, represented by the light-colored (orange) elements, is part of the private environment of an application provider. The white and gray arrows reflect the interfaces that should comply with the CUSS standard.

The CUSS environment comprises these elements:

CUSS Platform:

The kiosk CUSS Platform is the main element of a platform provider's environment. It consists of the Application manager, the System Manager Interface, the Common Launch Application (not shown in diagram) and the device components.

Kiosk Application
The kiosk applications runs inside or outside the kiosk and communicates with the CUSS platform (Application Manager and Device Components) via the CUSS interface.

AL Application Server(s) or Host(s)
Most kiosk applications might do backend communication with servers or hosts in their application provider's environment (e.g. airline check-in applications will communicate with an airline's check-in system).

AL System Manager
The airline application provider uses his management platform to retrieve system management information from the kiosk. It communicates with the kiosk CUSS platform (System Manager Interface and Device Components) via the CUSS interface.

SP System Manager
The service provider uses his management platform to manage the kiosks and their device components. If a platform provider runs his kiosks on his own, he is his own service provider. It also communicates with the kiosk CUSS platform (System Manager Interface and Device Components) via the CUSS interface

PP Distribution Server(s)
The PP distribution server, controlled by the platform provider, contains all the software components (application (if any) and platform) that should be loaded onto the kiosk.

1.7 Data Security Considerations

Many CUSS kiosks are deployed with CUSS application that process payment transactions using magnetic stripe cards inserted by the customer. In addition, the applications may support other magnetic card operations such as Identification and Loyalty Account updates.

Because these magnetic stripe card operations allow the kiosk user to insert a Payment Card (whether or not a payment card is requested), the CUSS platform and CUSS applications must ensure that sensitive payment card information is not exposed as part of magnetic stripe card processing.

In this document, “sensitive data” is considered to be any payment information that confirms with *ISO/IEC 7813 Identification cards -- Financial transaction cards*, including data read not only from magnetic cards, but any other current or future media that provides the same data.

Throughout this document, any references to “sensitive data” or “magnetic payment track data” should this be interpreted as referring to any ISO7813-compliant data, regardless of the media source.

In the current CUSS Technical Specification 1.3, platforms must follow the considerations outlined below as a condition of reading sensitive data from magnetic stripe payment cards.

Note: IATA intends that as part of an industry move to next generation payment solutions, future releases of the CUSS Technical Specification, will eventually remove the capability of reading complete magnetic payment cards completely to reduce the risk of exposing payment card information.

Note: Aside from this section, the CUSS Technical Standard does not address any security considerations for receiving, providing, handling, or forwarding private or sensitive data. It does not qualify any aspect of the standard or interfaces as containing sensitive data.

The CUSS-TS is a technical interface that only defines how kiosk devices are controlled and how data from those devices can be exchanged - it does not prescribe how the data is used or must be protected using technical controls.

It is the responsibility of CUSS platform and application vendors/suppliers, providers, and integrators to be aware of all data security and data privacy standards that are in effect at their locations, and they must ensure that all components in a CUSS kiosk solution under their control abide by these applicable standards.

Because these standards may change independently of the CUSS Technical Specification, and vary from location to location, the CUSS standard does not enumerate these applicable data security standards nor does it list the data handling guidelines that are be needed to meet their requirements.

In addition, the CUSS Technical Specification does not replace, supersede, enforce or guarantee any separate data handling and processing standards that may apply. Compliance with the CUSS Technical Specification does not imply or convey compliance with any other data handling or processing standards, or security guidelines.

As a general principle, CUSS platform and application providers must acknowledge that CUSS kiosks provide access to this sensitive data and take steps so that this data:

- Must be solicited from the kiosk user only when needed and in accordance with card scheme operating regulations
- Must never be logged or stored at the kiosk
- Must only be sent to known and trusted entities with a “need to know”
- Must be protected if sent over a network

Application and platform providers may wish to deploy Data Loss Prevention and similar system data monitoring/audit tools to assist in meeting these requirements and reducing exposure risk.

Here are some specific requirements that all CUSS platforms and applications shall follow:

1.7.1 Requirements for CUSS platforms

1. Platforms must implement the CUSS FOID Addendum (Chapter 8) on all CUSS components that provide application access to magnetic stripe card data. This means that payment card data is sent only to applications that explicitly request full payment data for use as part of a payment transaction.

2. Platforms must not log or store (encrypted or otherwise) sensitive data on the kiosk. Sensitive data may be logged only if it is truncated in accordance with applicable standards.

3. Platform providers must ensure that lower tier tools deployed on their kiosk, such as OEM device control toolkits, 3rd party software libraries, or serial/USB control tools, do not expose sensitive data.

4. Platforms must not maintain sensitive information in memory or session storage beyond the immediate time frame needed for the CUSS application requests. For example, platforms shall purge card data from their buffers after the application calls receive() to get the data.

5. Platforms shall transmit sensitive data over a network only if there exists a business need to do so. Any network traffic that contains sensitive data must be encrypted using appropriate and compliant encryption methodologies.

1.7.2 Requirements for CUSS applications

1. CUSS applications must abide by the CUSS FOID Addendum (Chapter 8) and shall request full payment card track only as part of a payment transaction. For all other functions, such as passenger identification, applications must not request full payment card data from the platform. This requirement is Subject to certain exceptions outlined in Chapter 8.

2. CUSS applications must not log or store (encrypted or otherwise) sensitive data on the kiosk. Sensitive data may be logged only if it is truncated in accordance with applicable standards.

3. Application providers must ensure that lower tier tools deployed on their kiosk, such as application programming toolkits, generic/rebranded components, 3rd party software libraries, and CUSS interface components, do not expose sensitive data.

4. Applications must not maintain sensitive information in memory or session storage beyond the immediate time frame needed for the customer payment transaction. For example, applications shall purge card data from their buffers as soon as the payment transaction is complete.

   Applications shall transmit sensitive data over a network only if there exists a business need to do so. Any network traffic that contains sensitive data must be encrypted using appropriate and compliant encryption methodologies.

5. CUSS applications should only transmit sensitive data as required by their business rules and application architecture. For example, if an application transaction server only needs the payment account number to complete a payment transaction, the application should only send the PAN and should not send complete track data.

6. While most CUSS applications run locally, the CUSS CORBA Interface Definition allows an application to connect to CUSS platform components “over the wire” from a remote server. Any CUSS application that connects “over the wire” to the CUSS interfaces using CORBA from a central application server, should redesign the card handling logic to ensure that all processing of sensitive data takes place at the kiosk. For example, a local “stub” applet should call receive() from the kiosk and encrypt the data being sent to the server, even though all other CORBA communication is directly from the server.

Ch 2: Interface Overview

This section describes the interfaces between a CUSS Application/CUSS System Manager and a CUSS platform, as shown in the complete CUSS Environment in Section 1.6. This section will cover the following:

Interface Communication Layer

Application Architecture Options

Interface Directives and Events

Application Manager Interface (AMI)

System Manager Interface (SMI)

Device Component Interface

2.1 Interface Communication Layer

Standard CORBA IIOP will be used between all ORB communications. This applies to the communication between any CUSS Application/CUSS System Manager and a CUSS Platform. The underlying communication layer between any CUSS element inside the kiosk and any other program outside the kiosk is based on TCP/IP (Figure 6 shows all of the elements in the complete CUSS environment). For example, Application Client communication between the kiosk and the Application Server or Host is done over an IP network.
To enforce security in communications, the application that wants to encrypt the data to/from a CUSS component must have an Application Client that includes code that is capable of encryption. The encryption of the data is not part of the CUSS standard. An application that does not want to use the encryption facility, may access the CUSS component directly from an external platform.
Network security is beyond CUSS standard and requires an SLA between the platform provider and the Airline application provider (e.g. network VPN between airline kiosk application and its application server).

2.1.1 Local vs. Remote Interface Connections

This section is taken from CUSS 1.0 Addendum A.1.5.

An airline application provider can request the use of a network-reachable address or hostname to communicate with the platform, in which case the kiosk must provide it. This can be a requirement if any part of the application logic is run remotely, for example on an application server.

However, the localhost interface is a valid kiosk IP and it is strongly recommended that applications running locally on the kiosk use this address (for connections, and callback interfaces.) This avoids many problems with dynamic DHCP lease renewal, media sense/disconnect on physical interfaces, etc.

If interfaces other than the localhost (127.0.0.1) interface are available on the kiosk, the kiosk provider must inform the airline application providers if there are any restrictions regarding which interfaces can be used by a particular airline application. This can be an issue if multiple, possibly airline-specific, network interfaces are available.

All system management and component interfaces must be accessible of the network, so that remote management can occur as intended by the CUSS specification.

2.1.2 CORBA TCP/IP ports used by CUSS platform and applications

This section is taken from CUSS 1.0 Addendum A.1.29.

By default, CORBA ORBs often allocate ports to interface objects from the entire available range of ports. When these CORBA objects communicate over a LAN or WAN, it is difficult to configure and manage firewall and access rules within the network, as these rules often restrict traffic by port number.

To allow CUSS applications and the platform to interact via CORBA object over a remote network connection, the Interface Communication objects (virtual components, event listeners) must use ports from a fixed, predefined range of TCP/IP ports. This allows network, kiosk and firewall administrators to more easily configure and restrict the communication between the kiosks and the kiosk/airline networks.

To allow this, the CUSS platform shall create all interface objects so that they listen on TCP/IP ports in the range 20000-20199 (as described elsewhere in this document, standard reference ports 20000 and 20001 are already define; other ports may be used for items such as callback listeners, device component interfaces, etc.)

Likewise, CUSS applications that are written to run remotely (where the CUSS objects are running on an application server, not on the local kiosk) shall be written to use port(s) in this range as well, for event listeners.

Consult your CORBA ORB technical documentation to understand how to create interface objects which use ports in this range.

This section applies only to the CUSS interface CORBA objects used by CUSS platforms and applications. It does not apply to or restrict the ports that can be used by an application to communicate with its backend hosts and servers. Any TCP/IP network access that an application needs to access its remote servers should be discussed and documented by a kiosk provider as part of the application integration and airport deployment

2.2 Application Architecture Options

The CUSS environment provides several options for creating the architecture for an Airline Application. This is represented in three basic categories (refer to Figure 7):

Multi-Tier Client - In a Multi-Tier application a portion of the application is resident on the CUSS kiosk and the remainder resides on the remote server(s). For instance, the presentation layer may be resident on the CUSS kiosk and the business logic may reside on the remote server(s).

Fat Client - The application may be deployed as a fat client where the application is loaded locally on the CUSS kiosk and provides its business logic and presentation logic locally.

Thin Client - In the case of a thin client application, the business logic and presentation logic, both reside on a remote Host/Application Server. The access to this application is via a browser running on the CUSS kiosk.

In all of the above cases, the interface to the CUSS platform remains the same. This is achieved by accessing the Application Manager CORBA Object reference by using CORBALOC (refer to Section 3.4) with the pre-configured IP address or host name of the CUSS kiosk. This will be true whether the call is made locally on the CUSS kiosk or from a remote server.

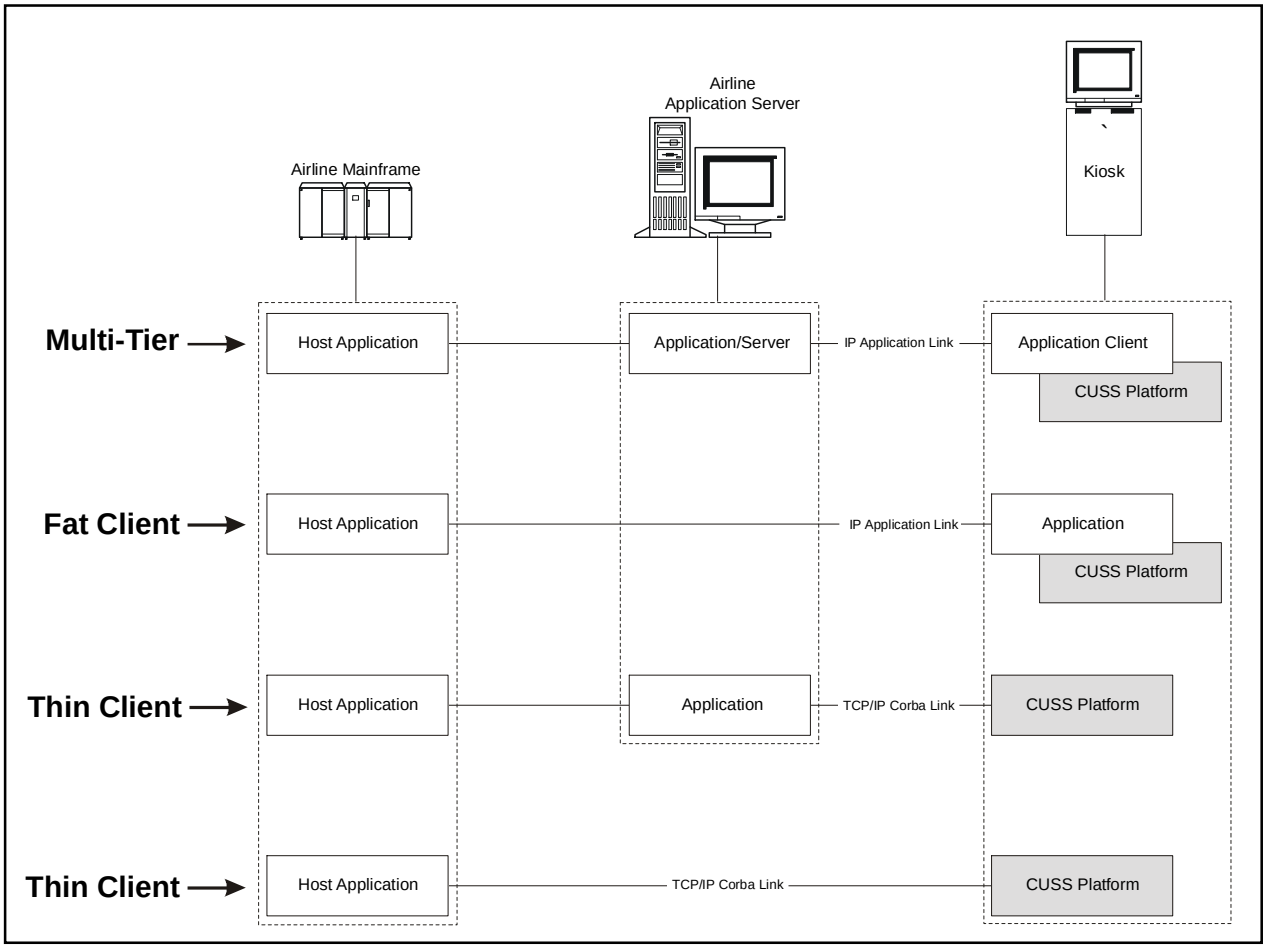


Figure 7 Application Architecture Options

2.3 Interface Directives and Events

Interface Directives and Events are provided to allow applications to access the platform services such as access to peripherals and communication with the Application Manager. Directives and Events also provide flexibility to the application; depending on its needs, the application can access the platform services using only directives or a combination of directives and events.
The design of Directives and Events is generic in nature in order to make the interfaces applicable to different types of device components, including any new ones that may be added in the future. For instance, sending data to a SmartCard or to a generic printer will use the same function call. It is important to note however that this will add a level of complexity to the application and platform development.
The usage of Directives and Events will be restricted by the platform based on the appropriate application state, device state, and rights assigned to the application.

2.3.1 Directives

Directives are high-level functions/methods for CUSS Application Manager/System Manager Interface and Device Component Interfaces to perform specific actions on a platform component on behalf of an application.
To allow flexibility for the application architecture, directives can be called in two **modes**:

Synchronous: the interface implementation will block the call until its execution is done or when timeout occurs. The result of the directive will be attached with the return call itself.

Asynchronous: the interface implementation will check the lexico-syntax of the calls and returns immediately. Only if the call is valid (i.e. directive is accepted), the information (or result of the action) requested will be returned via an associate event upon complete execution of the call.

Directives are divided in two categories in the CUSS standard:

Shared: can be used by any non-suspended AL application (e.g. generate an event for System Manager, put the application in UNAVAILABLE state, etc.), or by SP/ AL system managers.

Exclusive: can be used only by an active AL application (e.g. print, read, etc.) or by the SP system manager only if the kiosk is not in use (i.e., after suspending or stopping all kiosk applications). An AL System manager is not allowed to call an Exclusive directive.

No Directives can be used by suspended applications.
Note: All directives can be accessed locally or remotely. That is to say that all Directives are available for portions of the application running locally on the kiosk or on the application server/host. For more details on the directives and their definitions, refer to Section Ch 3:.

2.3.2 Events

The platform communicates with the application (AL or SM) via callback events regardless of the application state.
The main elements that constitute an event, including event listener mechanism and event processing, will be described in the following sections. For the actual definition of the event structure, please refer to Section 3.1.11.

2.3.2.1 Event Cause

An event can be caused by:

Hardware malfunction
Software malfunction
Acknowledgement of an existing error
Error repaired
Any normal situation change that can modify the self-service application or System Manager behavior
Asynchronous/synchronous interface call completion or abortion of a directive

2.3.2.2 Event Source

An event can be generated by:
Any application program sending events to a System Manager
CUSS Application Manager
Device Virtual Component

2.3.2.3 Event Modes

By definition, events are asynchronous, but they can be triggered in two modes:
Solicited: An event can result from an accepted asynchronous call to a directive.
Unsolicited: An event that is the consequence of a change in state of a component or an application. The event is not related to any previous directive call.

In either case, the information given by the event is the same as it is with the synchronous interface call of a directive. In the case of a component change, Unsolicited Events will be the same as the one returned to a Query (refer to Section 3.6.4) call to the same component.

2.3.2.4 Event Categories

Events can be divided in three categories:
Normal: normal processing occurs, not a detected error
Alert: abnormal situation occurs, but manual intervention is not needed
Alarm: requires immediate attention (i.e. manual intervention is required)

All alert and alarm events must be sent to System Manager software as they appear. The following chart shows the main distinctions of an alarm versus an alert:

Alert	Alarm
Status returns to normal automatically if the problem is solved or disappears	Status stays in alarm condition until acknowledged by SP System Manager and resolved by a human being.
Follows severity setting for problem solving	Requires an **immediate** response by a human being
e.g.: paper low, out of paper, device not reachable	e.g. kiosk door is open, temperature sensor is too hot or too cold

Figure 8 Alert Vs. Alarm

It is up to the platform provider and service provider to agree via an SLA on which events are classified as alerts vs. alarms. For example, a platform provider may choose to consider a device not reachable or out of paper as alarms while others may consider them as alerts.

2.3.2.5 Event Types

Events will be divided into four types (this is the publisher filtering):

Public: all AL applications and System Manager applications may receive the event.

Private: only the associated AL application and the AL System Manager (solicited event) may receive the event.

Platform: only the SP System Manager, CUSS Application Manager, CLA, and the CUSS Component Interfaces may receive the event.

Invalid: if a directive was called in asynchronous mode or a synchronous call was rejected, the returned event type should always be invalid.

All events MUST have at least one of the above-described types. The actual type is context dependant; e.g., Status OK can be either private or public: private if the status is given for an interface call from an application, public if the status is given from a change in the component state that just recovered from an abnormal status.

2.3.2.6 Event Codes

An event code reflects either an application or component state transition (or the actual state itself in case there is no state transition). For a list of all event codes, please refer to Appendix A.

2.3.2.7 Status Codes

A status code is part of the event definition. It describes the current status of a component or the result of the semantical analysis of a component interface call or the execution result of a component interface call. For a list of all status codes, please refer to Appendix A.

2.3.2.8 Event Listener Mechanism

The AL application and SP/AL System Manager may use only one general listener (please refer to the **`registerEvent`** directive in Section 3.3.5) and/or component specific listeners (please

refer to the **acquire** directive in Section 3.6.1) to receive all events: solicited events (the result of an asynchronous call) and unsolicited events (the result of a state change for which the application has done nothing).
When the application uses the **registerEvent** directive, with subscribe action, all events for which the application has subscribed will be sent to the listener associated to that directive. This allows an application to have only one listener.
When the event subscription is done at the Component **acquire** directive, all events, for which the application has subscribed, generated from the interface and underlying layer of that component, will be sent to the listener associated to that component. This allows an application to have one listener per component.
Nothing would prevent an application from using both mechanisms, allowing it to have centralized event processing when required and decentralized processing when it better suits the application architecture.
An event will be sent only to one listener of the application. In the event of a conflict in event subscription, the last registration to a specific event code of a specific component (regardless of the directive used and/or the type of event list used) will be used to find listener that will receive the event.
Subscriber filtering will also be implemented. The subscriber can choose the event that he wants to listen to by component object reference, by component type or by event category. For the complete event filtering definition, please refer to Section 3.1.12 (Event List Selection).
The application could change its subscribed filtering related to (or completely unregister) its general listener (via the **registerEvent** directive with discard option and the appropriate filter) or discard its component listener (via the Component Release directive, defined in Section 3.6.5).

2.3.2.9 Event Processing

Event must be processed as follows:

- All events must be sent, as they appear, to the subscribed application(s) according to the event type.
- All event and state modifications must be logged by SP System Manager component.
- It is the responsibility of the application to register the events and track component status.

Figure 9 represents the overview of event exchange:

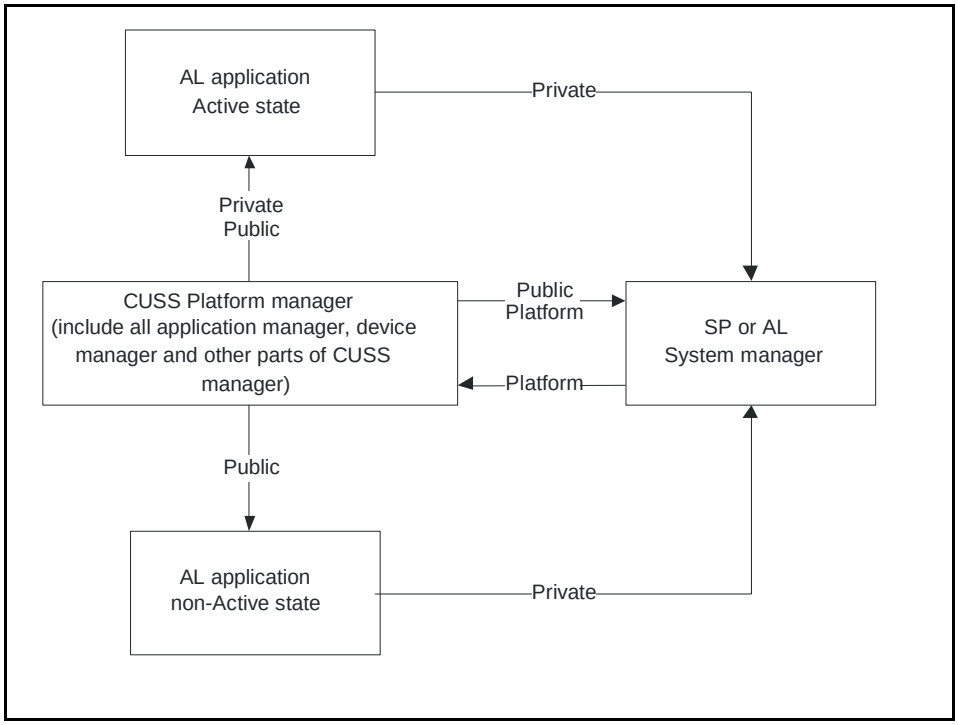


Figure 9 Overview of Event Exchange

2.4 Application Manager Interface (AMI)

The Application Manager Interface (**AMI**) defines all available functions to an AL application to access the platform services provided by the application manager. This includes moving an AL application from one state to another upon a request from the application, AL or SP system manager or the application manager itself. This section describes all the application states and all state transitions. For actual definitions for all AMI directives, refer to Section 3.2.

2.4.1 Application State Descriptions

This section describes all the states an AL application can be. Each individual state description includes how an application could have reached that state, what the application can and cannot do in this state and what are the possible next states. The states are listed in specific order to show a typical application initialization and activation sequence. An application state diagram that shows all the states and state transitions is illustrated in Section 2.4.2.

2.4.1.1 STOPPED State

The application is not loaded on the CUSS kiosk.

Only the CUSS application manager will load the application either automatically at system startup (e.g. kiosk boot time) or upon request of the SP system manager or its own AL system manager.

2.4.1.2 INITIALIZE State

This is the first state of the application when the CUSS Application Manager starts it:

The application manger loads an application using the pre-configured path. In case of a thin client, the application manager loads a browser with the pre-configured application URL.

The application starts executing when it gets loaded.

Only the CUSS Application Manager can load the AL application (either automatically at boot time or upon request of SP system manager or AL system manager).

CUSS Application Manager will allow an application to start initializing only in the following two situations:

When the system reboots.

When the Common Launch Application is executing (No AL application is active).

The first thing the application must do is to issue the Environment **`level`** directive to gets its token.

Next, the application must call **`registerEvent`**[4] to register its callback object so that the CUSS Application Manager can send events to the application.

Then, the application must issue a **`initrequest`** directive to inform CUSS application Manager that the application wants to enter the INITIALIZE state.

CUSS Application Manager is responsible to manage the initialization critical path. Only one application can be in INITIALIZE state at any one time. To enforce this, CUSS Application Manager will return the **`initrequest`** function call issued by the application only when the application is allowed to starts its Initialization

When the **`initrequest`** function call is returned, the application must check the event code and its associated status code.

If the application did not receive the proper event code with the 0 (OK) status code, the application must stop.

If the application enters the INITIALIZE state, it is allowed to perform the same actions as if it were in the ACTIVE state with the following exceptions:

No screen display

No printout

No reading

The application will then call the Environment **`components`** directive to get the list of components that exist on the kiosk, their characteristics and how they are linked among each other.

When the application terminates its initialization, it is its responsibility to use the **`notify`** directive to inform CUSS Application Manager that the phase is completed and the application enters in:

UNAVAILABLE state, for normal completion

STOPPED state, when the application terminates its execution.

[4] As **`initrequest`** and **`registerEvent`** are two independent synchronous directives, they can be issued in any order.

CUSS Application Manager can also put the application in DISABLED state, if it has a 'misbehavior'.

An application cannot be suspended while in INITIALIZE state.

An application can only use the initrequest() to enter the INITIALIZE state. The initrequest() directive is a blocking call and supplants the notify() directive for this state transition. The CUSS platform should return RC_DENIED and not grant the initialization request if an application calls notify() with the STOPPED_INITIALIZE transition. (From CUSS 1.0 Addendum A.1.31.)

2.4.1.3 UNAVAILABLE State

The application has completed its initialization process and is now ready to check its environment before putting itself in the AVAILABLE state:

The application uses the **`notify`** directive to inform CUSS Application Manager of its change of state either when it leaves the INITIALIZE state or the AVAILABLE state to enter into the UNAVAILABLE state.

The application still executes in background to monitor its environment requirements.

If the application found that the environment becomes proper for a correct execution in the ACTIVE state, it is its responsibility to send an event to the Application Manager (via **`notify`** directive) to inform that it wants to be in the AVAILABLE state

The application can also issue a **`notify`** directive to inform CUSS Application Manager to stop its execution.

The application is still able to communicate with the AL host/server.

The application can handle all subscribed events that the application can receive.

The application can still access all virtual components via shared directives only

CUSS Application Manager can also put the application in the DISABLED state, if it has misbehavior.

CUSS Application Manager can also put the application in SUSPENDED state upon request from AL System Manager or SP System Manager for operational reasons.

2.4.1.4 AVAILABLE State

The application has satisfactorily completed its environment check process and wait to be selected while still checking its environment:

The application uses the **`notify`** directive to inform CUSS Application Manager of its change of state either when it leaves the UNAVAILABLE state or the ACTIVE state to enter in AVAILABLE state

The application still executes in background to monitor its environment requirement.

If the application found that the environment becomes improper for a correct execution when activated, it is its responsibility to request the application manager to put it in the UNAVAILABLE state by issuing a **`notify`** directive.

The application can also issue a **`notify`** directive to inform CUSS Application Manager to stop its execution.

The application is still able to communicate with the AL host/server.

The application can handle all subscribed events that the application can receive.

The application can still access all virtual components via shared directives only.

Normally the application is waiting to be selected by a user (by receiving an event from Application Manager to become active).

CUSS Application Manager can also put the application in DISABLED state, if it has misbehavior.

CUSS Application Manager can also put the application in SUSPENDED state upon request from AL System Manager or SP System Manager for operational reasons.

2.4.1.5 ACTIVE State

A passenger has asked to use the application (its displayed icon on the CLA was selected):

The application is notified of its change of state by an event from application manager.

The application can access all virtual components via both shared and exclusive directives that are not restricted to applications in INITIALIZE State.

When the application completes its session (e.g. completes passenger check-in), it is its responsibility to use the **`notify`** directive to request CUSS Application Manager to move it to:

AVAILABLE state, for normal completion, or

STOPPED state, when the application terminates its execution.

CUSS Application Manager can also put the application in DISABLED state, if it has misbehavior.

An application cannot be suspended while it is in ACTIVE state. This is to allow the application to complete its session and correctly serve its users.

2.4.1.6 SUSPENDED State

The application was suspended for any operational reason upon request of SP system manager or its own AL system manager:

The application is notified of its state change by an event from application manager.

The application cannot use any component, it is only allowed to communicate with its server/host and listen for events.

A manual/automated intervention is required to change the application state via the application manager, upon request from SP/AL system manager, to either

STOPPED state: the application will have to be reloaded

Previous state: the application resumes execution in the state it was suspended from

An application can be suspended twice (first by SP SM and then by its own AL SM or vice-versa).

If system restarts, the application will be reloaded (i.e. will not remain suspended).

2.4.1.7 DISABLED State

If the application had an incorrect behavior, the CUSS Application Manager will move it to the DISABLED state:

The application is notified of its state change by an event from application manager.
All acquired components by the application are released by CUSS platform.
The application execution is stopped and it is unloaded.
A manual intervention is required to change the application state via the CUSS Application Manager or the SP System Manager to
STOPPED state: the application will have to be reloaded at next system startup, or
INITIALIZE state: the application will restart its execution.
An application cannot be suspended while in DISABLED state (if suspended, it is possible to be automatically reloaded at next system startup without human intervention).
If system restarts, the application will NOT be reloaded without a prior human intervention.

2.4.2 Application State Diagram

The Application state diagram (see Figure 10) illustrates how an application can move from one state to another. The application itself, the service provider system manager, or the application provider system manager can request an application state change. These changes are accompanied by an event sent to the corresponding application by the CUSS Application manager either as an unsolicited event or as a returned event upon a **`notify`** directive called by the application itself. Note that the numbers shown on the state transitions reflect the corresponding event codes. Refer to Section 3.7.3 for more information on these events. State transitions represented in solid thick lines means that a human intervention is required for this state transition to occur.

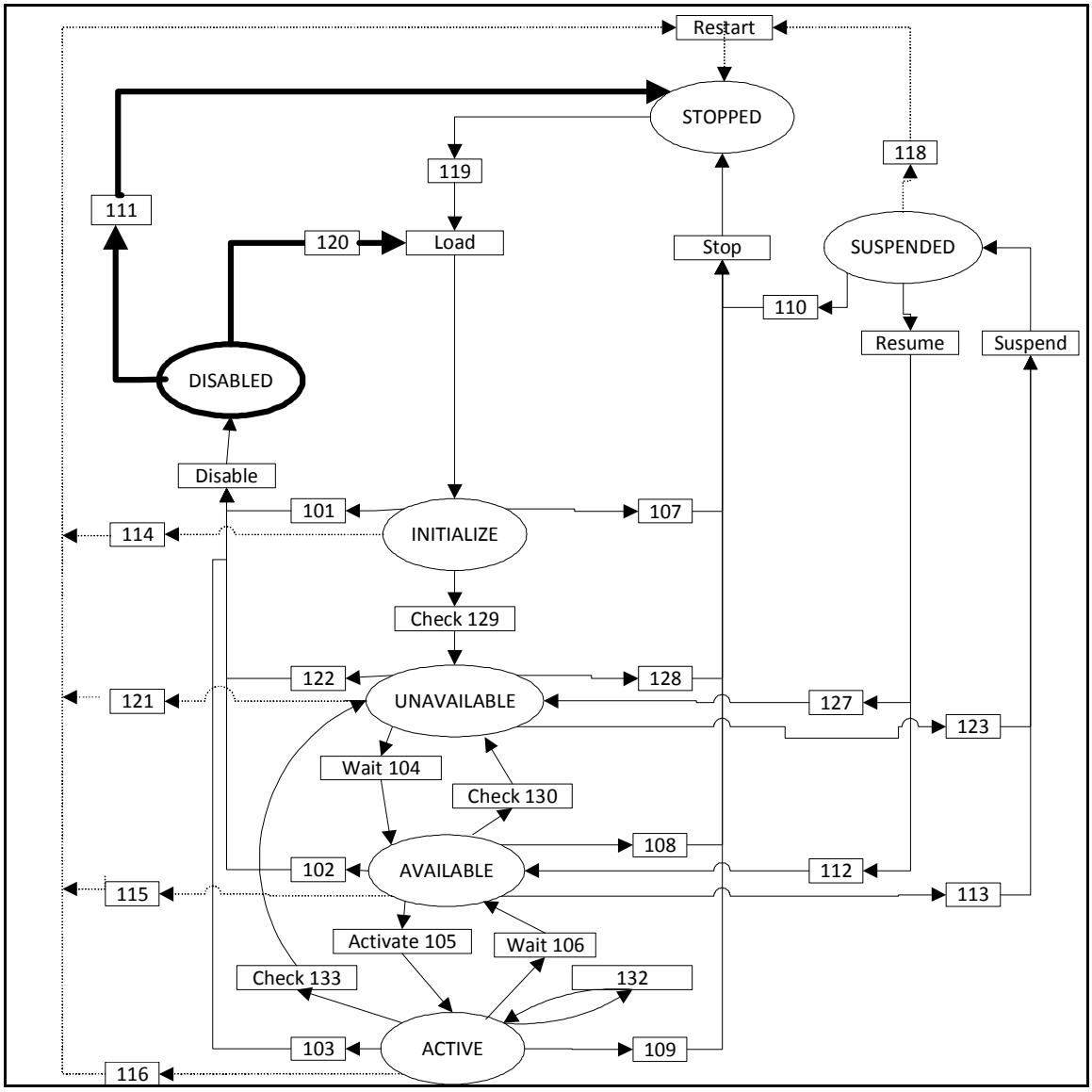


Figure 10 Application State Diagram

2.4.3 Application State Transition Description

2.4.3.1 Load Transition (STOPPED to INITIALIZE or DISABLED to INITIALIZE)

Used to load or reload an application in the system:

CUSS Application Manager loads an application upon request from its own AL system manager or the SP system manager via the load directive (refer to Section 3.5.1) or

CUSS Application Manager loads an application upon system startup or

CUSS Application Manager loads a disabled application upon human intervention.

The application will enter in INITIALIZE state when permitted by application manager.

While an application is initializing, the Common Launch Application will display a "Temporarily not available" type of screen.

2.4.3.2 Check Transition (INITIALIZE to UNAVAILABLE or AVAILABLE to UNAVAILABLE or ACTIVE to UNAVAILABLE)

The application has either completed its initialization or has found out that the CUSS environment is not suitable for its proper execution. Therefore, application requests application manager to be moved into UNAVAILABLE state.

The Common Launch Application will either remove the application icon from the screen or display it as un-selectable[5].

2.4.3.3 Wait Transition (UNAVAILABLE to AVAILABLE or ACTIVE to AVAILABLE)

The application has determined that the CUSS environment is adequate to its proper execution or it has completed its session. Therefore, it requests to be moved to AVAILABLE state.

The Common Launch Application will display the application icon as selectable.

2.4.3.4 Activate Transition (AVAILABLE to ACTIVE or ACTIVE to ACTIVE)

A user has selected the application and it starts its session.

The CUSS Application Manager upon request of the Common Launch application controls this state transition. The application manager will put the application window into the foreground.

The Common Launch Application continues to display the application icon.

2.4.3.5 Suspend Transition (to SUPSENDED)

The application execution is suspended:

CUSS Application Manager controls this state transition upon request of the SP System Manager or the application AL system manager.

The Common Launch Application will either remove the application icon from the screen or display it as un-selectable [5]. If this was the last icon to be removed (i.e. this is the last application to be suspended), the Common Launch Application should display "kiosk not in service" type of screen.

2.4.3.6 Resume Transition (back to pre-suspended state)

The application is now allowed to be operational:

Only the System Manager (SP or AL) that has suspended the application can apply this state transition.

The application will return to the previous state (the state in which the application was before to be suspended).

If both system managers have suspended the application, (SP and AL), it needs to be resumed by both of them before returning to its previous state (this is to resolve potential conflict within operational rules of SP System Manager and AL System Manager).

The Common Launch Application will display the application icon as selectable if the resultant state is AVAILABLE, and will either remove the application icon from the

[5] To remove a button or make it un-selectable is a decision left to SLA agreement between platform provider and application provider. It is recommended that the platform will make this option configurable.

screen or display it as un-selectable[5] if the resultant application state is UNAVAILABLE or SUSPENDED.

2.4.3.7 Disable Transition (to DISABLED)

Used to disable an application (put it into penalty box) until human intervention occurs:

CUSS Application Manager will put the application in DISABLED state because of an incorrect behavior such as:
Session time limit exceeded
Application threshold error exceeded
Etc.

CUSS Application Manager stops the application execution (i.e. unloads it).

The Common Launch Application will either remove the application icon from the screen or display it as un-selectable[5].

2.4.3.8 Stop Transition (to STOPPED)

Used to stop an application execution:

CUSS Application Manager put the application in STOPPED state upon request from the AL application itself, its own AL system manager or the SP system manager.

CUSS Application Manager stops the application execution (i.e. unloads it).

The Common Launch Application will either remove the application icon from the screen or display it as un-selectable[5]. If this was the last icon to be removed (i.e. this is the last application to be stopped), the Common Launch Application should display "kiosk not in service" type of screen.

2.4.3.9 Restart Transition

Rebooting the kiosk or restarting the CUSS platform activates this transition.

CUSS Application manager will put all applications, except those who were in DISABLED state, in STOPPED state and start loading them. See Load Transition.

2.4.3.10 Periodic/Automatic restart of the application

If a kiosk application runs for a very long time (days or weeks) on a busy kiosk, it is possible that the environment in which is running is susceptible to memory leaks or other resource problems that are not directly caused by the CUSS application. This is particularly a problem with browser-based applications– after running for days and thousands of page transitions, the browser uses too many resources and can affect the operation of the entire system.

As Internet Explorer is in widespread use, and it is the cause of many of the resource issues for browser-based applications, CUSS 1.2 makes the following recommendation for platform and applications. Because CUSS applications cannot restart themselves:

1. CUSS platforms should implement a feature under which applications running on a kiosk are restarted on a regular basis. This could be done by time of day, day of week, a certain number of transactions, by monitoring the resource usage of the process, or by other rules.

2. Applications provider will request that their application be restarted on a regular basis (nightly, etc.) as part of their deployment instructions, if required.

3. An application shall be able to run for an entire day on a busy kiosk, without requiring a restart (for example, 100-200 transactions.)

4. The intent of this Automatic Restart recommendation is only to address the problems inherent to the browser or java application "container" on the platform. Automatic restarting shall not be used as a substitute solution for poor application implementations: many resource leaks are due to improper application development and the application supplier shall attempt to correct these problems where possible.

Even though CUSS 1.3 specifies that IE8 be the default browser environment used on the kiosk, platform providers may choose to investigate and use alternate browser containers (that remain based on the IE8 rendering engine) for CUSS applications running on their kiosk.

These alternate browser containers might not exhibit the negative tendencies of the complete Internet Explorer product. (Some vendors already deploy a custom browser which, despite being based on Internet Explorer browser engine, does not exhibit the same resource problems as Internet Explorer.)

2.4.4 Modes of Operation for applications

A CUSS applications running on a kiosk can operate in one of three modes of operation: multi-application mode, single-application mode (SAM) and dedicated single-application mode. A separate application-transfer mode is also possible in combination with any of these three modes of operation.

These different modes allow a CUSS kiosk to run a different style of operation to meet the various needs of the airport and airline. For example, a kiosk running at a dedicated airline counter may operate in single-app mode, whereas a shared kiosk in a common area or parking garage will usually operate in multi-application mode. It is also possible that a CUSS kiosk changes from one mode of operation to another at various times of the day or week, without restarting the individual CUSS applications.

It is important that CUSS applications not be written to assume any particular mode of operation. In fact, a CUSS application does not need and should not implement any special logic to operate in either of these modes. Dedicated single-application mode, however, does require some additional application logic.

The decision about the mode in which an application is run on a kiosk is subject to the control of the CUSS kiosk provide and negotiation with the CUSS application providers. CUSS applications cannot “select” the mode they need at start-up.

2.4.4.1 Media-off-roller (MOR)

This section is taken from CUSS 1.0 Addendum A.1.5.

To support the various modes of operation, a new Media-off-Roller (MOR) concept is required. This concept allows certain key devices to be enabled and receive before any CUSS application is in the ACTIVE state, and defines the mechanism by which the next ACTIVE application can access that device information.

An application uses the same CUSS device component logic whether or not MOR is place on the kiosk. From the applications’ point of view, the kiosk devices behave identically whether or not Media-off-Roller is in use. As such, CUSS applications must not have any “special” handling for kiosk component beyond its normal behaviour.

For information on how a CUSS application handles device data and events, please see the complete documentation of device component behaviour below.

The operational goal of Media-off-Roller is to allow a CUSS kiosk to behave similarly to a legacy airline kiosk, where devices were enabled at all times. Where listed below, a CUSS 1.2 platform must support Media-off-Roller, and this support must be implemented in compliance with the following requirements:

1. MOR must be enabled only if an application requires it.
2. MOR must be turned on only for those devices that an application expects. For example, the card reader and barcode scanner may be used, but not the ATB2 coupon reader.
3. If the CUSS kiosk enables a device for MOR, and receives input from the kiosk user (such as a card read) then the CUSS platform must:

    a. Generate all device component events that would normally be generated for the same device behaviour for an ACTIVE application (these events may occur on one or more linked components, as appropriate.)
    b. Queue up all these events in the order they are generated.
    c. Activate the target airline application using the normal ACTIVE transition.
    d. Wait for the ACTIVE application to call the enable() directive.
    e. Broadcast to the application all queued events for that component.

For example, if Media-off-roller is used for a dip-style card reader, then the platform will enable the reader's MediaInput component prior to an application being active:

1. The kiosk user inserts a card
2. The platform generates and queues up the relevant components events such as MEDIA_PRESENT, DATA_PRESENT and MEDIA_ABSENT events
3. The CLA immediately activates a CUSS application
4. The ACTIVE application calls enable() on the MediaInput component as part of its activation logic.
5. The platform broadcasts all queued events (MEDIA_PRESENT, etc) for that component.
6. The application receives the events via its component listener, and processes the DATA_PRESENT as part of its business logic.

2.4.4.2 Multi-application Mode

This is the traditional mode of operation of a CUSS kiosk. In this mode, the Common Launch Application (CLA) presents an application menu with one or more airline selection buttons, and the correct CUSS application is activated.

A CUSS 1.0 compliant platform must be able to provide this mode of operation, as it is the normal assumed behaviour of a common-use kiosk.

A CUSS platform may allow Media-off-Roller in multi-application mode, but this is an optional feature. If MOR is enabled, a specific application (possibly based on the type of data received, for example a frequent flyer card, or based on platform configuration) is activated immediately.

2.4.4.3 Single-application Mode (with Common Launch)

This section is taken from CUSS Addendum A.1.5.

In this mode of operation, only a single CUSS application is running on the kiosk. When the application is not active, the Common Launch Application displays an “attract loop” that is controlled by the platform. When required, the CUSS application is activated immediately without displaying an airline selection menu.

A CUSS 1.2 compliant platform must be able to provide this mode of operation.

The CUSS platform must support Media-off-Roller in single-application mode. MOR must be configured and enabled for the devices needed by the application, if requested by the application.

2.4.4.4 Dedicated or Persistent Single-application Mode

This section is taken from CUSS Addendum A.1.10.

As an extension to Single-application Mode, Dedicated Single-application Mode allows a single CUSS application to be active at all times with its screen visible to the user, instead of the kiosk Common Launch Application. In this special mode of operation, the CUSS platform and application must implement some additional logic to ensure smooth operation.

A CUSS 1.2 compliant platform must be able to provide this mode of operation. This mode is sometimes called Persistent Single-application Mode.

When running in this mode, a compliant CUSS platform must:

1. Designate and run a single CUSS application in dedicated mode, in agreement with the application provider.
2. Transfer the application to ACTIVE state as soon as the application has reached the AVAILABLE state, without waiting for user input.
3. Include the notification string “SINGLEAPP MODE” in the ACTIVE transition (see below.)

4. Even though the application is active, do not start the SESSION and KILL timeouts until:
5. Any component is enabled by the application
6. The application issued the ACTIVE_ACTIVE transition
7. The platform needs the application to revert to AVAILABLE for any reason (for example, to switch to another mode of operation.)
8. In all other aspects of operation, the platform shall behave in accordance with the full CUSS specification.

To be able to run in dedicated single-application mode, a compliant CUSS application must:

1. Detect and use the “SINGLEAPP MODE” notification string during the ACTIVE transition from the platform.
2. Issue the ACTIVE_ACTIVE transition as soon as a customer is detected.
3. Transition to AVAILABLE if and only if a real transaction is finished. It must NOT perform this transition as a result of a screen timeout on the initial screen if a customer is not present.
4. Detect and process the SESSION_TIMEOUT event if it receives one from the platform.
5. In all other aspects of operation, the platform shall behave in accordance with the full CUSS specification.

The platform shall consider a transaction started, and start the SESSION and KILL timeout calculation, whenever any of the following conditions occurs first:

1. The application indicates ACTIVE_ACTIVE as an explicit start to the transaction
2. The platform broadcases MEDIA_PRESENT or DATA_PRESENT to the application, for any component.
3. There have been ten (10) touches on the touchscreen since the application was activated.

Media-off-Roller is not used in dedicated single-application mode, as the application is always active. The CUSS platform must support Media-off-Roller in single-application mode. MOR must be configured and enabled for the devices needed by the application, if requested by the application.

2.4.4.5 Application Transfer Mode

This section is taken from CUSS Addendum A.1.50.

The Application Transfer Mode allows trusted applications to pass control in the ACTIVE state from one application to another automatically, instead of requiring user input on the Common Launch menu. This mode of activity applies in multi-application mode, single-application mode, and even dedicated single-application mode.

Its primary purpose is to allow multiple CUSS applications to coordinate their transactions, as is sometimes needed for proper handling of code-share flights, irregular or alliance operations, or other situations where a passenger cannot be processed within the application they selected.

This mode also allows a new class of CUSS applications, “intelligent” programs that identify which actual CUSS application is the correct one to process a passenger. For example, a ground handler, consolidator or airport could create an application that determines which charter operator, alliance partner, or airline application is required to process a passenger, based on such parameters as time-of-day, destination, flight number, frequent flyer card, etc.

A CUSS 1.2 compliant platform must be able to provide this mode of operation, but it must be turned off by default for all CUSS applications. This mode can be combined in single-application mode (dedicated or not) as well as multi-application mode.

To implement this mode of operation, a CUSS platform must:

1. Implement a control or configuration mechanism that explicitly allows specific application transfers, by application. For example, application A may be able to transfer to B, but not C or D, whereas application B is able to transfer to A and C. By default, transfers must not be allowed.
2. All other applications that could be activated by transfer must be running on the kiosk, even if the main application is running in single-application mode, or running in multi-application mode but without a Common Launch Application selection button.
3. Support the generateEvent() request with the structure listed below to identify transfer requests from applications.
4. Respond to the generateEvent() request with the correct response code as listed below.
5. If a transfer request is accepted, the platform must immediately activate the new application as soon as the current ACTIVE application (making the request) transitions to AVAILABLE state.
6. The ACTIVE transition message to the new application must include the transfer data as provider in the transfer request, verbatim, without any changes or additions.
7. Any subsequent transfer request from the current ACTIVE application cancels the existing transfer request, regardless of the contents of the generateEvent() event.

To transfer between themselves, CUSS applications must:

1. Create the correct generateEvent() invocation to request the transfer.

2. Include in that request any transfer data that needs to be sent to the new application.
3. Interpret the return code from this request in accordance with its business logic requirements.
4. Issue a second generateEvent() request if ever the transfer needs to be cancelled.
5. Transition to AVAILABLE to allow the platform to activate the new application.
6. Examine the ACTIVE notification event data to detect, extract and use any transfer data that was provided by the previous application.

Application Transfer generateEvent() contents

To request a transfer, an application must invoke and the CUSS platform must support the generateEvent() directive as follows:

appRef	CUSS application reference of the ACTIVE application making the transfer request
ie.eventCode	Numeric value ACTIVE_TRANSFER from codes.idl (1001.)
ie.kioskID	Kiosk ID structure of the app that the current active application wishes to transfer to. Specifically, the values for ie.kioskID.companyCode and ie.kioskID.applicationName must refer to the target application.
ie.eventData	Event data that the active application wants to transfer to the target application. This is an arbitrary "datastream" structure (CORBA any) so any information in any agreed-upon format can be transferred between applications. The platform stores and forwards the event data unmodified. To be consistent with the existing "activation notification" design of CUSS (see Addendum A.1.4) it is recommended that the transfer event data be a msgDataType array with one record whose value is a freeform string.

Application Transfer generateEvent() return code

A CUSS platform must respond to the application transfer event request using the appropriate response code as defined:

RC_NOT_SUPPORTED	Application transfer requests not configured/supported on this kiosk.
RC_REFERENCE	The appRef or ie.kioskID values refer to applications that aren't configured on the kiosk.
RC_STATE	The application making the request is not currently ACTIVE.
RC_UNAUTHORIZED	The application making the request is not allowed to transfer control to the requested target application. CUSS platforms should implement proper permissions control.
RC_SHARE	The request is canceling a previous transfer request made by the requesting application, which has not yet ended its session.
RC_ERROR	The target application is not in the AVAILABLE state, so cannot be activated. Applications cannot transfer control to themselves (to extend total session time, for example.)
RC_OK	The transfer request is granted and the platform will activate the target application as soon as the requesting application ends its session.
RC_PARAMETER	Existing response for CUSS 1.0 or 1.1 platforms that do not recognize

	code 1001.

2.4.4.6 Multiple Application Brands

This section is taken from CUSS Addendum A.1.4.

It is possible that a single CUSS application supports multiple “brands” as part of its business logic, to cater to different types or categories of customer. For example, some airlines have created distinct operating brands (regional flights vs. mainline flights) or some ground hander applications may handle operations for multiple airlines.

To allow an application to display the correct branding when it is active, a Common Launch application in multi-app mode can be configured with multiple buttons that activate the same application. To activate different brands, these buttons are configured to activate the same application but with a different notification string. These notification values are provided as part of the application.

A CUSS 1.2 compliant platform must be able to support multiple button brands in multi-application mode.

To implement this mode of operation, a CUSS platform must:

1. Support a configuration element that sets the brand notification string for each button on the Common Launch application menu.
2. When a button is selected, the brand notification string for that button is included in the ACTIVE transition event for the target application.

To support this mode of operation, a CUSS application must:

1. Provide to the kiosk administrator the list of brands and exact notification strings it supports.
2. Detect and process the brand notification string while processing the ACTIVE transition event.
3. Support the possibility that extra string data is provided during ACTIVE notification, in addition to the brand notification.

2.4.4.7 One Application Instance per Process

This section is taken from CUSS Addendum A.1.4.

In some cases, a single kiosk application may need to provide the user interface or logic for more than one airline. This often occurs for applications that provide LDCS self-service check-in at

the airport, but can be used in other areas such as airlines with distinct brands, and as the result of mergers.

On a CUSS kiosk, even if an application “represents” multiple airlines, it must only connect to the platform as a single application (via level(), initrequest(), etc.) In other words, the same system process cannot connect to the CUSS platform as two separate applications “A” and “B”, and this restriction applies to any child processes launched by the application. This restriction is to allow the kiosk platform to properly track and managed application processes running on the kiosk and the CUSS objects and resources assigned to those processes.

To use multiple brands, application providers should make use instead of the “ACTIVE Brand Notification” feature in CUSS 1.2, discussed in Section 2.4.5.2 below. If this is not sufficient, then separate instances of the same application code shall be launched as completely separate applications (using, for example, different command line parameters or start-up URLs to set the different behaviour.)

2.4.5 Special State Transitions and Notification Strings

To implement some of the new features in CUSS 1.2, the concept of "notification string" is added to the ACTIVE transition event, as well as a new transition to indicate the start of a customer transaction when running in dedicated single-application mode.

2.4.5.1 ACTIVE Transition Notification String

To pass information from the platform to the active application during the ACTIVE transition, the CUSS platform can embed string data in the notification event.

This notification string may be used to indicate multiple types of data (brand, language, mode of operation) so applications cannot depend on "exact matches" when processing the activating notification.

String notification is set by including the string as the first record of a msgDataType object inserted into the eventData field of the ACTIVE transition event, which is sent by the CUSS platform to the CUSS application event listener at activation time. If the application needs this data, the application must extract and analyze this string and take appropriate action as required by its business function.

A platform that provices an active notification string must include the prefix "NOTIFICATION=" before the notification string, to allow applications to quickly determine the correct notification data (as different from

2.4.5.2 ACTIVE Brand Notification

This section is taken from CUSS 1.0 Addendum A.1.4.

When an application is activated from a multi-application mode menu button which has been configured with a brand notification value (as supplied by the CUSS application provider during initial setup) this value is included in the activation notification string.

This data is added to, and does not replace, any other notification string data included by the platform during activation.

The application can use this value to change its look, feel, or behaviour as it sees fit.

If there is also an ACTIVE Transition Notification String, the Brand Notification is added after that string without any additional prefix.

If there is NOT also an ACTIVE Transition Notification String, the Brand Notification string must include the prefix “NOTIFICATION=” to allow applications to quickly identify the notification data.

2.4.5.3 ACTIVE Language Notification

This section is taken from CUSS 1.0 Addendum A.1.16.

If the Common Launch application supports its own language selection option on the selection menu or attract screen, it should notify the active application of which language was selected by the user. This can avoid an additional language selection within the application, which could frustrate the user.

The language notification string must be in the format “LANGUAGE=LanguageTag”. This LanguageTag must be a string that is compliant with IETF RFC3066 “Tags for the Identification of Language” such as “en”, “en-ca”, “fre-ca”, etc. See http://www.ietf.org/rfc/rfc3066.txt for more information.

This data is added to, and does not replace, any other notification string data included by the platform during activation.

The application can use this value to change its look, feel, or behaviour as it sees fit. Typically, it would be used to select and activate the appropriate language within the CUSS application (if supported.)

2.4.5.4 ACTIVE Dedicated Single-app Mode Notification

This section is taken from CUSS 1.0 Addendum A.1.10.

If an application is activated in Dedicated Single-application Mode (see above) then the CUSS platform must include the notification string “SINGLEAPP MODE” in the ACTIVE transition.

This data is added to, and does not replace, any other notification string data included by the platform during activation.

Because the application running in this mode must behave slightly differently, a kiosk shall only operate in this mode if it is known that the CUSS application supports this notification, as

indicated by the application provider. An application must then use this dedicated single-application mode notification string to implement the compliant behaviour (see above.)

ACTIVE_ACTIVE Transaction Start Message

This section is taken from CUSS 1.0 Addendum A.1.10.

An active application running in dedicated single-application mode must use the notify() directive while active to indicate to the platform that a customer transaction has begun. The numeric constant value to use is 132, which will be added to CODES.IDL as ACTIVE_ACTIVE in CUSS 2.0.

The following behaviour is required for CUSS 1.2 compliant platforms and applications. Previous versions shall return RC_PARAMETER, as would be the case for any other invalid/unknown code passed to the notify() directive.

1. This event code is only valid when called by an ACTIVE application running in dedicated single-app mode. In all other cases, the platform shall return RC_DENIED.
2. The numeric value of ACTIVE_ACTIVE is 132.
3. The session and kill timers for the current session shall only start when ACTIVE_ACTIVE is received.
4. The platform can issue SESSION_TIMEOUT at any time if ACTIVE_ACTIVE is not received.
5. If the application has already called this function in the current session, the platform shall return RC_DENIED on the second and subsequent calls.
6. Once ACTIVE_ACTIVE is called, the SESSION and KILL timeout events shall be generated as per the timeout values returned to the application during initialization (just like any regular non-SAM session.)
7. The platform can use ACTIVE_ACTIVE to accurately track application session times, etc.

2.4.5.5 ACTIVE Application Transfer Notification

This section is taken from CUSS 1.0 Addendum A.1.50.

If an application is activated in response to a transfer request from another application, the ACTIVE notification event must include in its eventData structure the exact data object provided by the original application in its request.

This data completely replaces any other notification data present in the activation. It is the application's responsibility to parse, analyze and use the transferred data as required by its business function. For example, this data could include passenger name or booking data, in a format that is known to both applications.

2.4.5.6 Application Status "Reason" Indicator

This section is taken from CUSS 1.0 Addendum A.1.42.

One challenge faced in maintaining common-use kiosks is to help the kiosk provider determine why the applications running on its kiosk are not operating normally (not AVAILABLE, for example.) This is an issue because applications have their own business logic and controls which may affect the availability of the application beyond simple conditions like device errors or paper supply problems.

So a goal is to provide a mechanism for a CUSS application to indicate simple, human-readable text messages that summarize why an application is in a particular state. For example, these could be something like: "Outside of service hours", "Kiosk device not found (ATB printer)", "Kiosk device out of service (passport reader)" or "Airline DCS Host not reachable".

Any such indication is completely optional, and the contents of the notification are up to the application provider. For example, an application does not need to reveal sensitive information like "DCS system too busy." An application can include such things as detailed error codes, URL links to monitoring pages, or any other information they choose.

It is up to the CUSS platform provider to implement a system management tool that then properly exposes this useful information, when provided by the applications, to the kiosk administrator. This information would usually be used for live monitoring and logging purposes, and would not be usually displayed to the kiosk user (such as on an out-of-service screen.)

The proposed mechanism is to use the generateEvent() directive with a custom eventCode value.

appRef	CUSS application reference of the application or system manager setting the reason text.
ie.eventCode	Numeric value STATE_EXPLANATION from codes.idl (1000.)
ie.kioskID	Kiosk ID structure of the application for which the reason text is being set. Specifically, the values for ie.kioskID.companyCode and ie.kioskID.applicationName must refer to the application. This allows airline system managers to set reason text for the applications it manages.
ie.eventData	Event data that includes the plain text reason as the first record in a msgDataType structure. Multiple records can be used (between application and its system managers) but only the first record will be used by the CUSS platform and management tools.

The return code from the platform shall be as follows:

RC_NOT_SUPPORTED	Application reason text method not supported by platform.
RC_REFERENCE	The appRef or ie.kioskID values refer to applications that aren't configured on the kiosk.

RC_UNAUTHORIZED	The application making the request is not allowed to set the reason text for the requested target application.
RC_OK	The request is granted and the platform maintains the reason text until the target application returns to AVAILABLE or ACTIVE state.
RC_PARAMETER	Existing response for platforms that do not recognize code 1000 (older platforms prior to CUSS 1.2.)

This event will be broadcast to the callback interfaces of the application itself and all service provider and application system managers for that application. The platform itself may choose to broadcast its own reason text event when setting the application to SUSPENDED, STOPPED or DISABLED.

2.4.5.7 Application Status "Transaction" Indicator

For the same reasons as listed in Section 2.4.5.6, it is useful to allow an application to report to the CUSS platform a description of the result of the latest transaction. This can also assist in analyzing the behaviour of a kiosk application, by indicating the result of the application's most recent transaction. For example, it could be "Passenger not found" or "Screen timeout" or "Too late for check-in." The message would be tied to the current (or most recent) time during which the application was ACTIVE

Any such indication is completely optional, and the contents of the notification are up to the application provider. For example, an application does not need to reveal sensitive information like individual passenger names or PNRs. An application can include such things as detailed error codes, URL links to monitoring pages, or any other information they choose.

It is up to the CUSS platform provider to implement a system management tool that then properly stores and exposes this information, when provided by the applications, to the kiosk administrator. This information would usually be used for live monitoring and logging purposes, and would not be usually displayed to the kiosk user (such as on an out-of-service screen.)

The proposed mechanism is to use the generateEvent() directive with a custom eventCode value.

appRef	CUSS application reference of the application or system manager setting the reason text.
ie.eventCode	Numeric value TRANSACTION_EXPLANATION from codes.idl (1002.)
ie.kioskID	Kiosk ID structure of the application for which the reason text is being set. Specifically, the values for ie.kioskID.companyCode and ie.kioskID.applicationName must refer to the application. This allows airline system managers to set reason text for the applications it manages.
ie.eventData	Event data that includes the plain text reason as the first record in a msgDataType structure. Multiple records can be used (between application and its system managers) but only the first record will be used by the CUSS platform and management tools.

The return code from the platform shall be as follows:

RC_NOT_SUPPORTED	Application reason text method not supported by platform.
RC_REFERENCE	The appRef or ie.kioskID values refer to applications that aren't configured on the kiosk.
RC_UNAUTHORIZED	The application making the request is not allowed to set the reason text for the requested target application.
RC_OK	The request is granted and the platform maintains the reason text until the target application returns to AVAILABLE or ACTIVE state.
RC_PARAMETER	Existing response for platforms that do not recognize code 1002 (older platforms prior to CUSS 1.2.)

This event will be broadcast to the callback interfaces of the application itself and all service provider and application system managers for that application. The platform itself may choose to broadcast its own reason text event when setting the application to SUSPENDED, STOPPED or DISABLED.

2.4.5.8 Automed Remote Update VERSION_EXPLANATION

CUSS 1.3 adds a new APPLICATION_VERSION indicator event that applications can generate at startup to report their current version to the platform, and to obtain Automated Remote Update parametes back from the platform.

Applications must generate this event at startup if they wish to perform Automated Remote Updates. Otherwise this request is optional for non-ARU applications.

The proposed mechanism is to use the generateEvent() directive with a custom eventCode value.

appRef	CUSS application reference of the application or system manager setting the reason text.
ie.eventCode	Numeric value VERSION_EXPLANATION from codes.idl (1003.)
ie.kioskID	Kiosk ID structure of the application for which the version text is being set. Specifically, the values for ie.kioskID.companyCode and ie.kioskID.applicationName must refer to the application. This allows airline system managers to set version text for the applications it manages.
ie.eventData	Event data that includes the plain text version string as the first record in a msgDataType structure. Multiple records can be used (between application and its system managers) but only the first record will be used by the CUSS platform and management tools as the version indicator.

The return code from the platform shall be as follows:

RC_NOT_SUPPORTED	Application version text method not supported by platform.
RC_REFERENCE	The appRef or ie.kioskID values refer to an application that is not configured on the kiosk.
RC_UNAUTHORIZED	The application making the request is not allowed to set the

	version text for the requested target application.
RC_OK	The request is granted and the platform records the specified version as needed for monitoring or ARU purposes.
RC_PARAMETER	Existing response for platforms that do not recognize code 1003 (older platforms prior to CUSS 1.3.)

This event will be broadcast to the callback interfaces of the application itself and all service provider and application system managers for that application.

The platform will provide an output even containing the parameters controlling automated remote updates the the application for which the version was specified. The output event will contain:

oe.eventCode	Numeric value VERSION_EXPLANATION from codes.idl (1003.)
oe.kioskID	Kiosk ID structure of the application for which the version text was set.
oe.eventData	The output event data will include a msgDataType structure with one or more records indicating the ARU parameters in effect for the application: • ARU time window in HHMM-HHMM format • ARU bandwidth limit in KB/sec • ARU suggested CPU limit in 0-100 percentage

In particular, the oe.eventData msgDataType response must be in this format:

- **msgDataType[0]** will contain “ARUTIME=HHMM,HHMM”
- **msgDataType[1]** will contain “ARUBANDWIDTH=xx”
- **msgDataType[2]** will contain “ARUCPU=xx”

Additional keywords may be added in the future, as needed to support the ARU business process.

2.4.5.9 Automated Remote Update UPDATE_REQUEST

CUSS 1.3 adds a new UPDATE_REQUEST event that applications must call before attempting to perform an Automated Remote Update (ARU).

The platform will respond whether the request for ARU is granted. An application that does not receive the correct response from this request must not use the ARU process for CUSS 1.3 described in Chapter 9.

The proposed mechanism is to use the generateEvent() directive with a custom eventCode value.

appRef	CUSS application reference of the application or system manager making the ARU request.

ie.eventCode	Numeric value UPDATE_REQUEST from codes.idl (1004.)
ie.kioskID	Kiosk ID structure of the application for which an automated update is requested. Specifically, the values for ie.kioskID.companyCode and ie.kioskID.applicationName must refer to the application. This allows airline system managers to request updates for the applications it manages.
ie.eventData	Event data that includes a msgDataType structure that includes information about the ARU request.

In particular, the ie.eventData msgDataType request includes information in this format:

- **msgDataType[0]** is required and must describe the version string of the "to be" version that would be in place after the update.

- **msgDataType[1]** is optional and can include any freeform information about the update as required. The platform may record this information for monitoring purposes.

The platform can make use of the following information to track and determine if the ARU operation requested is permitted:

- The kiosk identifier, station code
- The date and time of the request
- The applicationName and companyCode reported by the application
- The version reported by the application at start
- The version requested by the application as part of the ARU process

The return code from the platform shall be as follows:

RC_NOT_SUPPORTED	The application did not report VERSION_EXPLANATION at startup or did not set a "to-be" version in the request, or the versions indicated are not recognized but the platform's ARU oversight/notification process.
RC_REFERENCE	The appRef or ie.kioskID values refer to an application that is not configured on the kiosk.
RC_UNAUTHORIZED	The application making the request is not allowed to request an update for the requested target application.
RC_STATE	The ARU request was made outside the time window allowed for updates.
RC_SHARE	The platform has a local exemption in effect that is temporarily suspending ARU in this kiosk.
RC_OK	The request is granted and the platform records the specified version as needed for monitoring or ARU purposes.
RC_PARAMETER	Existing response for platforms that do not recognize code 1004 (older platforms prior to CUSS 1.3.)

This event will be broadcast to the callback interfaces of the application itself and all service provider and application system managers for that application.

Prior to returning RC_OK it is a platform responsibility to carry out any maintenance tasks needed to comply with the ARU Business Requirements described in Chapter 9. If the platform responds RC_OK to this request, the application may proceed with its ARU process, but is not required to do so.

If and once the application ARU process is complete, the application may request a process restart using the normal notify() requests transitioning to the STOP/RESTART state transition.

Upon startup, the application shall report its application version using the VERSION_EXPLANATION event described above. It is not a fault condition if the version reported by the application at this time is the same as before, because there are numerous conditions in which the application could restart prior to its ARU process being complete.

2.5 System Manager Interface (SMI)

CUSS provides the System Manager interface (SMI) to allow remote management of the CUSS kiosk environment. This interface allows access to the current state and status of all components and applications. It also provides the capability to manage AL applications and device components. This interface is available to the System Provider (SP) System Manager and Airline Provider (AL) System Manager. However, the System Manager Interface restricts access based on the current activity of the CUSS kiosk and the rights assigned to it. For instance, if there is an AL application active the SP System Manager cannot exercise any device components. As well, based on security policy, an AL System Manager cannot manage unauthorized AL applications. The Directives and Events applicable to SP and AL System Managers are detailed in Section Ch 3:.

SP/AL System Managers connect to SMI as their first CORBA object using CORBALOC as detailed in Section 3.4. The capabilities offered to SP and AL System Mangers are outlined in the following sections:

2.5.1 SP System Manager

The SP System Manager can *load*, *stop*, *suspend*, and *resume* AL applications. However certain functions may be limited by the CUSS platform. For example, if the AL System Manager suspends an application, the SP System Manager cannot *resume* it.

The SP System Manager can register its listener(s) to receive all events including alarms and alerts.

The SP System Manager is allowed to perform maintenance functions on device components when the CUSS kiosk is not in use. The SP System Manager must first *suspend* or *stop* all application during this operation, as it cannot relinquish control of the device components to any AL application. An AL application requires an exclusive control of device components when it is *active*. The maintenance could include such items as test prints or print head cleaning. This may vary depending on the type of device.

2.5.2 AL System Manager

AL System Manager can *load*, *stop*, *suspend*, and *resume* AL applications that are associated to AL System Manager by configuration. Again AL System Manager cannot *resume* an AL application that is *suspended* by SP System Manager.

AL System Manager can register its listener(s) to receive events, alarms and alerts however it will only receive public events and private events belonging to those AL applications that are associated to the AL System Manager by configuration.

AL System Manager is not allowed to perform maintenance functions on device components.

2.6 Device Component Interface (DCI)

The Device Component Interface is based on a defined virtual environment for any CUSS self-service application. This section illustrates this virtual environment including the description of all states virtual component can be and their associated state transitions. A component state diagram is also illustrated in Figure 12.

2.6.1 Virtual Component Concept

A specific CUSS implementation maps, on behalf of all CUSS self-service applications, the virtual environment against the real environment components. For example, a virtual receipt printer could be mapped to a real ATB2 device or a GPP device; a real printer with three bins with three different stock will be seen by the self-service application as three distinct printers, two real printer bins can be seen as one virtual bin allowing to use the stock of the second bin when the first one is out of stock, etc., all of which are controlled by the CUSS environment itself.

In addition, the CUSS component management concept includes virtual component chaining. For example, printer bins are seen as feeder component that can offer the required paper to a printer, which is a MediaOutput component. When the printing is completed, the document can be offered to a Dispenser component, which can be an escrow (if installed), and then offered to the user or to a Capture component (a capture bin - if installed)

All of these virtual components can be implemented in one real device or in many real devices. All virtual components related to the same real peripheral must have the same Real Component Name, allowing the AL application (if required) to know that fact. In the same manner, all linked virtual component will be cross linked in the virtual component table via the virtual component link table allowing the application to internally build the virtual component chaining without any requirement to know if these virtual components are implemented in one or many real devices

A real component (e.g. peripheral device) could be mapped to one or many virtual components. By defintion, there should one virtual component per real component per function per media type (the latter applies only for media-based peripherals). For instance, assume an ATB2 device with escrow supports coupon reading and revalidation, printing boarding passes/tickets/receipts, and capturing printed documents if the user fails to remove them. This ATB2 device will then be mapped to a number of virtual components as illustrated in Figure 11. The block diagram in Figure 11 also demonstrates the component chaining and linkage among the virtual components.

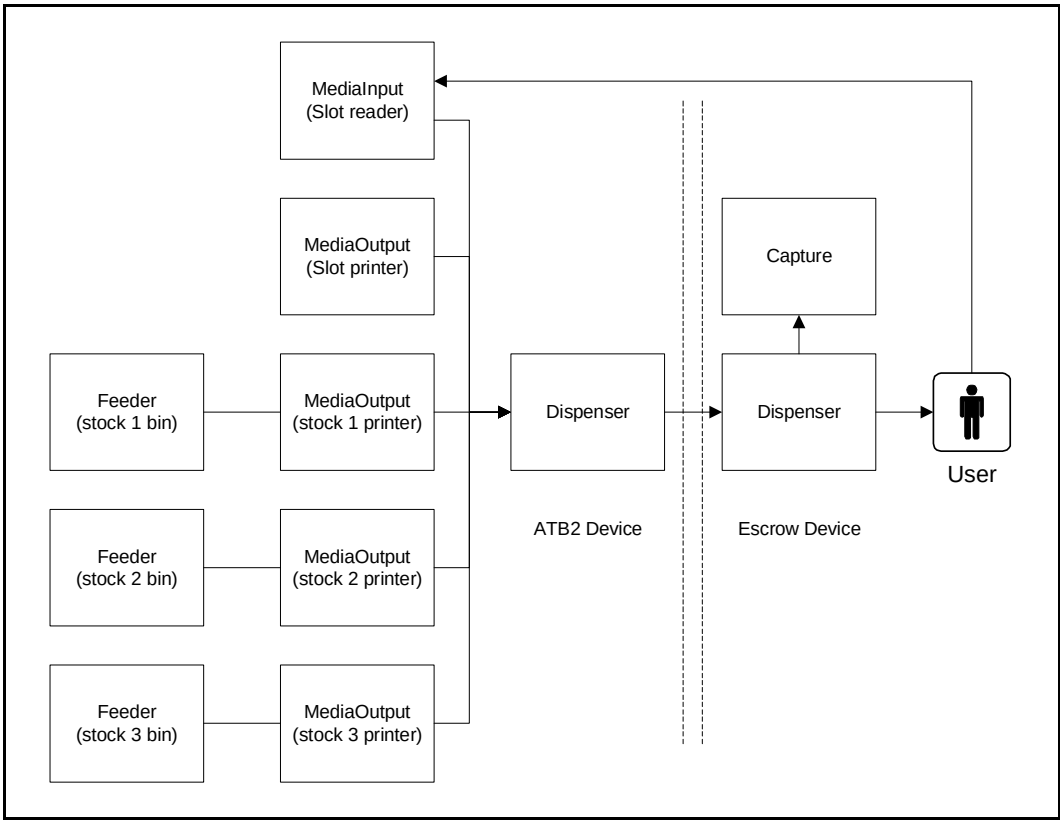


Figure 11 ATB2 Device with Escrow, 3 bins with distinct stocks

Another example is an ATB2 device with three bins that supports printing documents of two distinct types of stock. This device will be mapped to the following virtual components:

Feeder 1 associated with the first two real bins containing first stock type
Feeder 2 associate with the third real bin containing the second stock type
MediaOutput 1 (linked to Feeder 1)
MediaOutput 2 (linked to Feeder 2)
Dispenser (linked to MediaOutput 1 and MediaOutput 2)

Refer to Appx B:: Component Mappings to check how other real components are mapped to virtual components.
The mapping from virtual to real environment is done by CUSS entities: object, method (directive), etc. These entities are accessed by the self-service application via an interface call (refer to Section Ch 3:: Interface Definition). These interfaces use object-oriented programming techniques based on CORBA to implement functionality and events that define the CUSS work environment for self-service application.
Only one action on one single stock type is allowed in one directive on one virtual component at a time.
Please see Appendix D for information on printing multiple AEA documents (such as sequential bag tags) via a single request.

As the interface know exactly which function is used for which purpose on which component, the interface will be responsible for adjusting what is required to ensure the proper behavior of the interface call. For example, in case of the ATB2 device illustrated in Figure 11 above, assume MediaOutput 1 is a boarding pass printer linked to Feeder 1 that corresponds to Stock type 1 (Boarding Pass). If the AEA data stream sent to MediaOutput 1 specifies stock type 2, the implementation of the interface will adjust to make sure the data is printed on boarding pass document (stock type 1), since MediaOutput 1 is a boarding pass printer. On another cuss platform the boarding pass bin (Feeder linked to boarding Pass printer) may correspondent to stock type 3 instead of 1, then the interface will have to adjust the data stream to use bin 3 instead.
It is all based on the fact that virtual components are implemented as object networking handled by methods representing directives and events applying to real components and states. This allows changing any kiosk implementation without changing any, well designed, self-service application by modifying only the required CUSS interface implementation (underlying methods).
All virtual component have standardized characteristics for a given device type. The CUSS platform provides these characteristics to the application as well as allowing it to change some of them if necessary. Refer to Section Ch 5:: Virtual Component Characteristics for a list of characteristics per virtual component.

2.6.2 Some Device Component Interface Rules

All ATB and GPP printer interfaces whose virtual components are designated as boarding pass printers, must support AEA data stream (with exclusion of magnetic stripe for GPP). All virtual designated as bag tag printers must support AEA data stream
All GPP must support SVG/W3C standard formatting message.
Any document can be printed with:

On ATB2 device: using AEA data stream

On GPP printer: using AEA or SVG data stream; the former must be implemented only if no ATB printer is implemented in the kiosk, otherwise it is optional (for that kiosk)

Only ATB2 devices will use AEA data streams for reading.

Non-ATB readers (MediaInput) will use MSG data stream for reading.

Non-ATB printers (MediaOutput) will support a standard SVG data stream for printing. Each MediaOutput component is associated with a Dispenser component even if the real device does not have such real feature. In this case the printer output path is considered as being the dispenser. Since the Dispenser component is not real, the **offer** directive is not required to have the document delivered to the user.

The preceding paragraph applies also to MediaInput component that have to return media to the user.

Native commands must not be used by AL application and will not be supported by component interfaces.

Printer/reader memory management:

If 'n' ALs have an application on the printer/reader, each one will have 1/n of the printer/reader memory.

Each AL will have a minimum of 120KB of memory available (this gives 6 PECTAB of 4K, 4 templates of 2K, 8 logos of 10K)

If an AL application needs to download a PECTAB, a logo or any other file on the printer/reader and there is no more memory available, it is one of the requester AL file that will be offloaded from its context in the printer/reader.

An item will be downloaded only once, CUSS interface will cache every time the application does a download of the item and CUSS interface will retrieve the item if it is not loaded on the printer/reader when required by the application.

Character sets:

Related Documents

ISO/IEC 10646-1 second edition

Unicode 3.0.0 (http://www.unicode.org)

The Common Use Self Service (CUSS) standard was developed to provide a platform for application providers to write their applications once and to run that application on any kiosk system around the world that complies to the CUSS standard. To support the customers all other the world - especially those who are not familiar with the English language - or to conform to national standards, application providers must be able to write their applications in such a way, they can support different languages and different character sets. This do not imply that all devices of a platform has to support multiple languages or character sets only the ones that interact with the user have to support this. Because most of the data streams in the airline industry are ASCII-based, devices or their data stream that are already covered by another standard (IATA, AEA...) don't have to support other character sets than ASCII. But the operating system and the devices that are covered only by the CUSS standards have to support at least ISO 8859-1 (Latin 1) and double-bytes. Those who have to support other character set the use of Unicode 3.0.0 is mandatory. For compatibility reasons to ASCII the use of Unicode UTF-8 encoding is mandatory. UTF-16 can be converted to UTF-8 without lost of information.

When no application, other than CLA, is active, all components are available to the SP system manager only after suspending or stopping all applications.

2.6.3 Device Component State Description

Node Descriptions	
State	**Description**
RELEASED	This is the initial state of a device component. Within the ***RELEASED*** state, the component is ready to be acquired (for usage) by any application. Once a component is acquired by an application, it does not prevent other applications to acquire it at the same time. This means that multiple applications can acquire the same component(s) at the same time. Component **`query`** directive is NOT allowed in this state.
READY	The ***READY*** state tells the application(s), that the component is now ready to receive and execute any functions or directives given by a application. On receiving directives, the component changes to the ***BUSY*** state and remains there, until the function returns with either an error or OK. Only the application holding the active token will be able to execute any exclusive directives. All other applications will not have the permission to do so. An unsolicited event may lead from ***READY*** directly to ***EVENTHANDLING*** and then to UNAVAILABLE (e.g. manually switching off a printer). Releasing the component must be possible by either the application or any authorized platform component (e.g. when CUSS Application Manager disables the application).
BUSY	Completely transient state, which indicates any component activity (e.g. reading/writing) that has been invoked by an application or any authorized platform component. Calling component **`query`** directive in this state should return the last known status. All other directives will be queued if the request is valid.
EVENTHANDLING	Also a transient state. Defines that a component has to handle an event after detecting it. An event may be raised by a function invocation, an internal check resulting in a component condition, due to an unsolicited event or an exception (e.g. inserting a credit card or manually switching off a printer).

Node Descriptions	
State	**Description**
UNAVAILABLE	Defines an unrecoverable error condition that doesn't allow any function to be executed on the component. The events sent during the transition to this state allow applications to decide whether to be selectable on the launch screen or not. By receiving active error events from the real component device driver or carrying out internal checks or by human intervention (e.g. remove paper Jam), the component may become available on its own which may lead the component back into the ***READY*** state. UNAVAILABLE component was acquired before. Therefore, releasing the component in this state must be possible. Component **`query`** directive is allowed in this state.

2.6.4 Device Component State Diagram

The state diagram defines a common behavior of a CUSS component and not its implementation. But it defines exactly when an event has to be sent and what kind of event it has to be. It also defines the sequence in which events occur. The states, as viewed by the application, are used only to define transitions and events.
In this diagram, dotted ovals represent transient states. Also, the event codes numbering in this diagram is not related to the sequence in which the events may occur. The numbering is used only to identify the different events. Refer to Section 3.7.2 for more detailed information about these events.
The platform behavior will be the same regardless of what the application view of the virtual component state (e.g., if the application thinks a component is READY while it is actually UNAVAILABLE). This is to ensure that an application can use the state transition to know if a method call was successful or not successful.

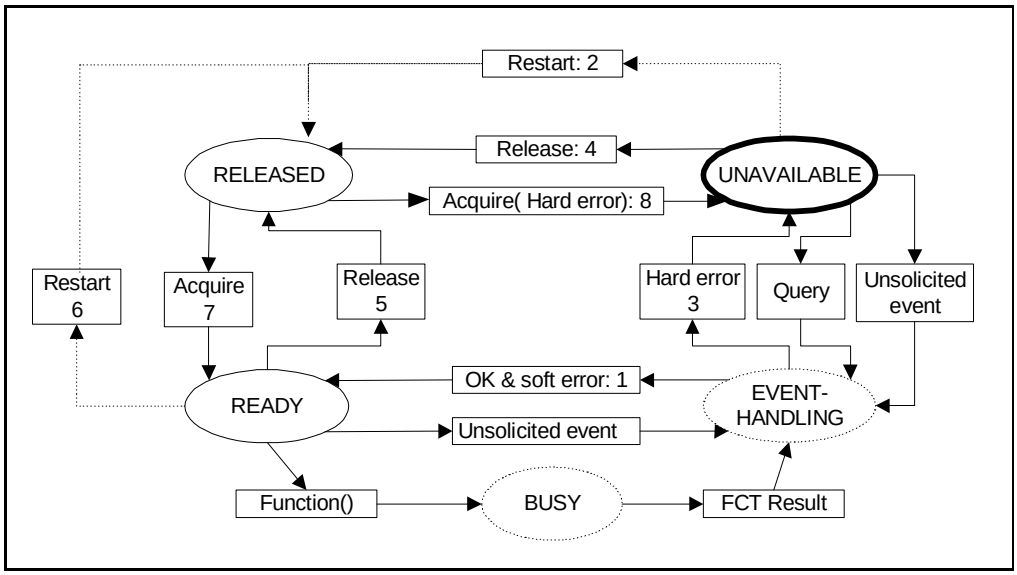


Figure 12 Device Component State Diagram (Application View)

2.6.5 Device Component State Transition Description

State Transitions	
Transition	**Description**
Acquire	A Component acquire directive is issued by the application. The component will either move to READY or UNAVAILABLE state.
Release	A Component release directive is issued by the application. The component will move into RELEASED state.
Function	This can be one of the following functions, depending on the component: **`retain, offer`** for media based components **`receive`** for input components **`send`** for output components **`enable`**, **`disable`** for user based components **`test`**, **`setup`**, **`cancel`**, for all components that inherit from class Peripheral **`query`** for all components. The component will either stay in READY state or move into UNAVAILABLE state.
Unsolicited Event	External events such as: card/coupon inserted, media jammed, device not reachable, device is now OK, printer out of paper, paper is low, paper is OK now, etc. Depending on the event, the resultant component state will be either READY or UNAVAILABLE.
Restart	Component is released by an authorized platform component. This may happened when the system restarts or when CUSS Application Manager moves an application to DISABLED state or when the application is stopped and its acquired virtual components are not released by the application itself. The component will move into RELEASED state.

Ch 3: Interface Definition

This section defines the CUSS Interfaces of the various elements of the CUSS system. It includes the following subsections:

Data Structures Definitions
Components Definition
Management Interface (MIF) Directives
Application Manager Interface (AMI) Directives
System Manager Interface (SMI) Directives
Device Component Interface (DCI) Directives
Event Listener Interface (ELI)
Media Device Behaviour and Event Sequence

AMI, **SMI**, and **DCI** are all based on directives issued by an application (AL or SM). **ELI** is the interface that the platform will use to communicate with an application (AL or SM). Refer to **Appendix C** for the actual CORBA IDL listings for all the data structures, components, directives and events.

3.1 Data Structures Definitions

This section presents data structures used at many places in the directives presented in this section. Field length and values are listed only when required for naming standard purposes.

3.1.1 Reference

It is an alias type for string used for application and component references.

3.1.2 Name

It is an alias type for string used for name definitions.

3.1.3 Timeout

Timeout: number <0: negative of the timeout value, asynchronous call
0: no timeout, synchronous (blocking forever) call
>0: timeout value; synchronous call
(Timeout values are expressed in milliseconds.)

3.1.4 Application Token

Application token: reference
(The CUSS Application Manager assigns this token to the application. It is used as an access control mechanism to directives available to applications)

3.1.5 Correlation

Correlation: any
(This is set by the application when issuing a directive (**`registerEvent`** or **`acquire`**) to register its listener, and it allows for the correlation of events to a specific directive issuance. It is to be used for comparison only as user-defined private identification.)

Note 1 (from CUSS 1.0 Addendum A.1.32):
CUSS application can insert whatever data they choose within this parameter to the acquire() and registerEvent() calls. Huge data objects should not be used, however, for performance reasons. Applications typically use a String or Long object as a correlation parameter.
The platform will do no data/type checking on this value, as it is application-specific, and the platform will return this same correlation data in each event broadcast to the event listener created in response to any of these calls. The application can then, if needed, analyze its correlation values as part of application event handling.

3.1.6 Vcomp Reference

VcompReference: reference

(This is the CORBA reference to the Virtual component (IOR).)

3.1.7 Kiosk Location

Location: Structure of { (name of the location of the kiosk)

Airport code: name (airport/location code, 3 characters)

Terminal: name (if applicable)

Area: name (if applicable)

Address: name (free form address, if applicable)

}

3.1.8 Kiosk GPS Coordinates

Coordinate: structure of { (coordinate of platform, international navy standard)

Longitude: structure of {

Orientation: set of {East, West, Undefined} (e.g. for Undefined is mobile kiosk)

Degree: number {0-179}

Minute: number {0-59}

Second: number {0-59}

Hundreds: number {0-99}

}

Latitude: structure of {

Orientation: set of {North, South, Undefined} (e.g. for Undefined is mobile kiosk)

Degree: number {0-89}

Minute: number {0-59}

Second: number {0-59}

Hundreds: number {0-99}

}

Altitude: number (in meters from sea level)

}

3.1.9 Data

Data[1]: structure of {

Data type: set of {AEA, CLOCK, SVG, SWITCH, MSG, NIL}

Datastream: case data type of {

AEA: character chain (message in AEA format)

[1] Please see Section 1.7 for guidelines on handling sensitive data, including card track data

CLOCK: character chain (format is "yyyymmddhhmmss")
(used by Clock device)

SVG: character chain (message in SVG format)

SWITCH: set of {OFF, ON, OPEN, CLOSED, YES, NO, UNKNOWN}
(used by sensor devices)

MSG[2]: Structure of {

Number of data records: number

Record [3]: Table [1. number of data records] of structure of {

Data Status: set of {OK, Corrupted, Incomplete, ZeroLength}

Message: character chain

}

)

NIL: nil

}

}

3.1.10 Kiosk Application ID

Kiosk-Application ID: structure of {

Application ID: structure of { (application to/for which the directive/event applies, leave blank if not related to a specific application)

AL code: name

(code associated to each airline for AL system manager, this value must be the same as the one passed via the Environment `level` directive, 3 characters)

Application name: name (unique name within AL code)

}

Kiosk ID: structure of {
(ID of the kiosk to which the directive/event applies, used for SM-Interface)

[2] If the media being read has a logical multi-track arrangement, then each track is returned as a separate msgData data "track". Examples of this include 2 or 3-track magnetic cards, 2-track standard passport MRZ data, 3-track National ID Card OCR data, etc. (Any valid multi-track document data can be received in this fashion. CUSS is not limited to only certain types of documents.)

3 In case of a card reader, each record corresponds to card track. If the track is formatted but empty then the record exists and the data status will be ZeroLength. If the track is not formatted then the corresponding record will not exist in the structure. Also, the message in each record should not contain any vendor-specific start and stop sentinel (e.g. "%", "?").

Vender code: name (vender code assigned at registration time, 3 characters)

Kiosk name: name (unique name within vendor code)

}

}

Note 1 (CUSS 1.0 Addendum A.1.30):
When used as an input parameter in a call to the platform (for example, for the level(0 call) the application ID values companyCode and applicationName are used to uniquely identify the CUSS application making the request. The vendorCode and kioskName parameters are ignored (ie, an application cannot request a specific Kiosk ID.)

The platform will populate the output EnvironmentLevel akID and location fields with whatever values the kiosk provider chooses provided they are otherwise CUSS-compliant.

An application vendor cannot assume that a specific kioskName is unique across its entire network. Only the combination of vendor code, kiosk location, and kioskName can be guaranteed unique.

An application vendor cannot require that a CUSS kiosk provide a specific kioskName. If an application requires a specific syntax or value of kioskName/location values for its operation, the application must include an internal mapping/lookup table or other feature that converts the platform-provided akID/location values into the format that the application requires.

3.1.11 Event

Event: structure of {

Timestamp: number of 100 nanoseconds in UTC format based on 15 October 1582 00:00 (event originator timestamp)

Kiosk-Application-ID: structure defined in 3.1.10

(ID of the kiosk application, if it is the event source or ID of the applicable kiosk application in case of the event source is CAM)

Kiosk Location: defined in Section 3.1.7 (kiosk Location (text format))

Kiosk GPS Location: defined in Section 3.1.8 (kiosk GPS Coordinates)

VCompReference: reference (virtual component reference if it is the event source)

Function: name (name of the directive invoked)

Event code: number defined in Appendix A (application/component state transition or the current application/component state if no transition applies)

Event mode: set of {SOLLICITED, UNSOLLICITED}

Event type: set of {PRIVATE, PUBLIC, PLATFORM, INVALID}

Event category: set of {ALERT, ALARM, NORMAL}

Status code: number defined in Appendix A (component or function call status)

Correlation: any (used for comparison only, set by application)

Data: structure defined in Section 3.1.9 (data passed with the event, if any)

}

Note: The same event structure is used regardless of the event source or cause. Therefore, some of the event fields may not always be applicable. In this case, the field value(s) may be left blank or their values are considered not applicable. The following are some examples on non-applicable values: kiosk-Application-ID if the event source is a device virtual component, VCompReference if the event source is a kiosk application, function name if the event is unsolicited, status code if the event source is a kiosk application.

3.1.12 Event List Selection[4]

Event List Selection: structure of {

Call type: set of {CODE, TYPE, COMPONENT, ANY, ALL, CATEGORY}

Call list: case call type of {

ANY: Nil (wait for any code of any component, used for **`waitEvent`** only)

ALL: Nil (apply to all codes for all component(s))

CODE: Event code selection: structure defined in Section 3.1.13

TYPE:Event type selection: structure defined in Section 3.1.14

COMPONENT: Component selection: structure defined Section 3.1.15

CATEGORY: Event category selection: structure defined in Section 3.1.16

}

}

3.1.13 Event Code Selection

Event code selection: Structure of {

Number of event codes: number

List1: table [1..number of event codes] of structure of {

Event code: number defined in Section0

[4] In **CUSS 1.0**, it is not required to implement event filtering. All Device Component events will be sent to listeners registered using the **`acquire`** directive. The event listener passed through **`registerEvent`** will be used for all events coming from CUSS Application Manager.

List type: set of {ALL, ANY, COMPONENT}

List2: case list type of {

ALL: nil (apply for this code to all components)

ANY: nil (apply for this code to any component, only used for **`waitEvent`** directive)

COMPONENT: Structure of {

(this list type can not be used with **`acquire`** directive)

Number of components: number

List3: table [1..number of components] of {

VCompReference: reference (virtual component IOR)

}

}

}

}

}

3.1.14 Event Type Selection

Event type selection: Structure of {

Number of event types: number

List1: table [1..**number of event types**] of structure of {

Event type: set of {PRIVATE, PUBLIC, PLATFORM}

List type: set of {ALL, ANY, COMPONENT}

List2: case list type of {

ALL: nil (apply for this code to all components)

ANY: nil (apply for this code to any component, only used for **`waitEvent`** directive)

COMPONENT: Structure of {

(this list type can not be used with **`acquire`** directive)

Number of components: number

List3: table [1..number of components] of {

VCompReference: reference (virtual component IOR)

}

}

}

}

}

3.1.15 Component Selection

Component selection: Structure of {

Number of components: number

List1: Table [1..**number of components**] of structure of {

Reference value: reference (virtual component reference)

List type: set of (ALL, ANY, CODE, TYPE, CATEGORY)

List2: case list type of {

ALL: nil (apply for all codes of this component)

ANY: nil (apply for any code of this component, used only for `waitEvent` directive)

CODE: structure of {

(Event code, for all status codes associated to the event listed)

Number of codes: number)

List3: table [1..number of codes] of {

CODE: number defined in Appendix A

TYPE: set of {PRIVATE, PUBLIC, PLATFORM}

CATEGORY: set of {NORMAL, ALERT, ALARM}

}

}

}

}

}

3.1.16 Event category Selection

Event category selection: Structure of {

Number of event categories: number

List1: table [1..number of event categories] of structure of {

Event category: set of {NORMAL, ALERT, ALARM}

List type: set of {ALL, ANY, COMPONENT}

List2: case list type of {
ALL: nil (apply for this code to all components)

ANY: nil (apply for this code to any component, only used for **`waitEvent`** directive)

COMPONENT: Structure of {
(this list type can not be used with **`acquire`** directive)

Number of components: number
List3: table [1..number of components] of {
VCompReference: reference (virtual component IOR)

}

}

}

}

}

3.2 Components Definition

A virtual component is defined with a set of classes, which have been divided into hierarchic classes. This section defines these classes and how each virtual component is mapped to these classes.

3.2.1 Component Classes

Component class	Description
Component	All parts that compose a CUSS kiosk platform. All components are derived from this class.
Component	
ManagementInterface	Component under control of CUSS Application Manager or System Manager Interface. Refer to Section 3.3.
CUSSCntl	Component under control of CUSS Device Components. Refer to Section 3.6.
CUSSCntl	
NativeDevice	Device used by the CUSS environment. They can be accessed, without the use of CUSS interface, only if mentioned in this chapter (like: disk, screen, network....). All of these native device must generate events to inform application and system managers about their status and availability
ApplicationComponent	Component used to query the state and/or characteristics of a kiosk application that is configured on the platform. (not to be confused with the actual application)
Peripheral	Input/output devices that are able to generate events to be sent to applications.
ManagementInterface	
ApplicationManager	Component that controls all kiosk applications on the platform. Refer to Section 3.4.

Component class	Description
SystemManagerInterface	Component that implements the SP/AL System Manager Interface. Refer to Section 3.5.
Peripheral	
Input	Components that provide data to applications
Output	Components that are able to receive data from applications
User	Components that interact directly with customers/users. See Note below
Userless	Components that don't interact with customers/users
Media	Components that use a physical media (e.g. card, coupon, or a paper document)
Medialess	Components that don't use a physical media (e.g. card, coupon, or a paper document)
Data	Components that transfer data
Dataless	Components that don't transfer data

Another way of reading the same table is:

- Component
  - ManagementInterface
    - ApplicationManager
    - SystemManagerInterface
  - CUSSCntl
    - NativeDevice
    - ApplicationComponent
    - Peripheral
      - Input
      - Output
      - User
      - Userless
      - Media
      - Medialess
      - Data
      - Dataless

Note: A component will be considered pertaining to the User class (UserInput or UserOutput virtual component type) if the answer to one of the following questions is yes:

Does the user have to interfere with the device in any way to make the data available?

Could it be useful for an application to put the device in disable status for any reason?

3.2.2 Virtual Component Definitions

A virtual component inherits attributes, directives and events from all classes that composed this component. For example, a UserInput virtual component is made of classes: User, Medialess, Data and Input; therefore, this virtual component is able to handle everything defined to these classes or their super-classes. In addition, a virtual component is composed of the real

component and its CUSS interface. For example, a MediaInput is composed of a real card reader (hardware), the card reader supplier driver (software) and the associate CUSS interface.
The table below shows the component classes each virtual CUSSCtnl component is composed of:

Virtual Components versus Component Classes	
Virtual Component Name	**Component Classes**
Application	ApplicationComponent
Capture (components that are able retain media)	Userless + Media + Dataless
DataInput (components used for inbound data transfer (e.g. digital input))	Userless + Medialess + Data + Input
DataOutput (components used for outbound user data transfer (e.g. screen))	Userless + Medialess + Data + Output
Dispenser (components that receive media from a Peripheral component and offer it to the user or to another Peripheral component e.g. ejecting an ATB coupon from the printer to the escrow)	User + Media + Dataless
Display (e.g. kiosk computer screen)	NativeDevice
Feeder (components that are holding media (e.g. ATB stock) and supply it to another Peripheral component)	Userless + Media + Dataless
LoggingServices	Not defined in CUSS 1.0
MediaInput (components used for reading from media (e.g. mag card reader)	User + Media + Data + Input
MediaOutput (components used for writing to media (e.g. receipt printer))	User + Media + Data + Output
Network (components handling network access)	NativeDevice
Storage (components used for reading/writing from/to storage (e.g. hard disk))	NativeDevice
UserInput (components used for inbound user data transfer (e.g. sound device))	User + Medialess + Data + Input
UserOutput (components used for outbound user data transfer (e.g. screen))	User + Medialess + Data + Output

3.2.3 Components that depend on a linked Component

For some devices, an action on one component may not be completed due only to an error in a linked component. For example, a MediaOutput component might not be able to print if a linked Dispenser component that is full of documents, even though the MediaOutput component has no errors.

If a component directive is called that depends on a linked component that is in a state that does not allow the directive to complete, that directive will fail with HARDWARE_ERROR or other failure status code (depending on the condition of the linked component) but the component on which the directive was called will remain AVAILABLE.

For example, if printing on a MediaOutput component when the linked Dispenser component is at MEDIA_FULL and cannot accept more coupons, the send() request would fail with MEDIA_FULL but the MediaOutput component would remain available.

3.3 Management Interface (MIF) Directives

MIF directives are shared by both the Application Manager Interface and the System Manager Interface. They are divided into two categories: **Environment directives**, which allow kiosk applications and the SP/AL System Managers to get a high level knowledge of the platform environment and the **Event directives,** which are related to event handling. Both Environment directives and Event directives are implemented using synchronous mode only.
Environment directives are the first two: level and components.
Event directives are: generateEvent, queryEvent, registerEvent, and waitEvent.

3.3.1 level

Description	This is the first directive that should be issued by a kiosk application or a system manager to get basic information on the specific CUSS Platform implementation. The calling application can validate the environment to check knowing if it can execute properly into this specific environment implementation. If the application is known by the platform (via platform configuration), the application reference (token) is returned with this call.
Apply to	ApplicationManager class
Available to	AL application in **INITIALIZE** state Service Provider System Manager Application Provider System Manager
Access	Shared, local/remote, synchronous
Structure sent	**Application ID[5]:** part of Kiosk Application ID, structure defined in Section 3.1.10
Code returned	**Function Return Code:** defined in Appendix A

[5] Only the Application ID part of the Kiosk Application ID must be filled by the application with the same information provided to CUSS Application Manager by configuration. The Kiosk ID part could be left blank.

Structure returned	Structure of {
	SessionTimeout: number in milliseconds (timeout value for an application session. Session is the period when an application is active. SESSION_TIMEOUT event will be sent when this timeout elapses)
	KillTimeout[6]: number in millseconds *(Time left before application is killed (moved to DISABLED state). KillTimeout starts when SessionTimeout expires. KILL_TIMEOUT event will be sent after KillTimeout elapses*.)
	
	Kiosk Location: structure defined in Section 3.1.7
	GPSLocation: Kiosk GPS Coordinates, structure defined in Section 3.1.8
	Kiosk ID[7]: part of Kiosk Application ID structure, defined in Section 3.1.10
	CUSS version: name contains a comma-separated string for all CUSS versions supported
	CUSS interface version supported Minimum level: name
	CUSS interface version supported Maximum level: name
	JVM name: name Name of the JAVA virtual machine used
	JVM version: name Version of the JAVA virtual machine used
	Browser name: name Name of the installed internet browser
	Browser version: name Version of the installed internet browser
	OS name: name Name of the installed operating system
	OS version: name Version of the installed operating system
	
	Application token: defined in Section 3.1.4
	}

3.3.1.1 Platform Version Information

For CUSS 1.2, the platform will include "1.0,1.1,1.2" in the **cussVersion** environment component. Platforms that support earlier versions will not include "1.2" in this string. The **cussInterfaceVersionMin** field will contain "1.0" or be blank.

For logging and troubleshooting purposes, the platform will set the **cussInterfaceVersionMax** field of the EnvironmentLevel structure to be a platform-specific, free-form string. This string will accurately reflect the internal (proprietary) version of the CUSS platform on which the application is running.

This string should not be used to make the behaviour of the CUSS application different for various platform vendors. Instead, for example, it could be used by an application to determine if the platform version is older or newer than the version against which the vendor performed integration testing.

[6] The minimum value for KillTimeout is 60000 (1 minute) to allow applications sufficient time to exit
[7] Only the kiosk ID part of the Kiosk Application ID structure need to be filled by the application manager.

3.3.1.2 Platform Location Information

Some CUSS kiosks might be located at areas other than airports such as convention centres. In some cases, this could allow kiosks serving multiple departure airports (for example, London or New York.)

A kiosk that is not deployed at an airport will contain the word "OFFSITE" within the address field of the EnvironmentLevel location. A CUSS application can use this indication along with the airportCode airport code (or possibly city code) to determine if a particular kiosk is offsite, and adjust its business logic if needed.

A kiosk service provider that is deploying kiosks offsite must communicate this information to airlines whose applications are running on those offsite kiosks. This is to allow airlines to opt off of these kiosks (for legal, regulatory, or other reasons.) if needed, airline providers can adapt their applications to support the offsite and airport code indicators to adjust the business logic of their CUSS applications.

3.3.2 components

Description	This is the second directive to be issued by an application allowing it to get a list of all implemented virtual components, their characteristics and their CORBA object references. This will allow the application to check whether all components that it requires are implemented or not.
Apply to	ApplicationManager class
Available to	AL application in INITIALIZE or ACTIVE state Service Provider System Manager Application Provider System Manager
Access	Shared, local/remote, synchronous
Structure sent	**Application Token:** defined in Section 3.1.4
Code returned	**Function Return Code:** defined in Appendix A

Structure returned	Table [0..**number of virtual components-1**] of structure of {
	Virtual component name: name (defined in Section 3.2.2
	Virtual component object reference: reference; (component IOR)
	Real component name: name (this is unique for a specific peripheral that is mapped to many virtual components, must be used for comparison only)
	LinkedComponents: structure of (
	Link table: Table [0..**number of links-1**] of {
	Virtual Component Table Index: number {from 0 to number of virtual component-1} (bi-directional direct component link only)
	}
	}

Note:
Component Characteristics could be accessed via the virtual component object reference.
All the callers will get a complete list of all NativeDevice and Peripheral components plus:

list of all ApplicationComponent components related to all configured AL applications on the platform if the caller is the SP System Manager

list of all ApplicationComponent components for a specific AL if the caller is the associated AL System Manager.

its own ApplicationComponent component if the caller is an AL application

3.3.3 generateEvent

Description	Generate an event to a System Manager. Application should not use **`generateEvent`** to communicate with Application Manager. It should **`notify`** instead.
Apply to	ManagementInterface Class
Available to	AL application in INITIALIZE, UNAVAILABLE, AVAILABLE, or ACTIVE state Service Provider System Manager Application Provider System Manager
Access	Shared, local/remote, synchronous
Structure sent	**Application token:** defined in Section 3.1.4 **Event**: to be generated: structure defined in Section 3.1.11
Code returned	**Function Return Code:** defined in Section A.1
Structure returned	**Event**: structure defined in Section 3.1.11

3.3.4 queryEvent[8]

Description	Return the description of the event(s).
Apply to	ApplicationManager Class All acquired virtual components of class "CUSSCntl"
Available to	AL application Service Provider System Manager Application Provider System Manager
Access	Shared, local/remote, synchronous

Structure sent	Structure of {
	Application Token: defined in Section 3.1.4
	Event list selection: structure defined in Section 3.1.12
	}

Code returned	**Function Return Code:** defined in A.1

Structure returned	Structure of {
	List type: set of {CODE, COMPONENT, TYPE} (list type returned will be the same as the one in the request)
	List1: Case **list type** of {
	CODE, TYPE: structure of {
	Number of codes: number
	List: Table of [1..**number of codes**] of structure of {
	Event code: number
	Event type: set of {PRIVATE, PUBLIC, PLATFORM}
	Event description: chain of characters
	Number of components: *number* (components to which the code applies)
	List2: Table [1..**number of components**] of name (virtual component name)
	}
	}
	COMPONENT: structure of {
	Number of components: number
	List: structure of {
	Component name: name *(virtual component name)*
	Number of codes: number

[8] In CUSS 1.0, the implementation of **`queryEvent`** directive is optional. If not implemented, it should always return RC_NOT_SUPPORTED as the function return code.

	List2: Table of [1..**number of codes**] of structure of (**Event code**: number **Event type**: set of {PRIVATE, PUBLIC, PLATFORM} **Event description**: chain of characters } } } } }

3.3.5 registerEvent

Description	Subscribe to or discards from receiving any related event notification. The use of this directive has an additive effect, i.e. a call will not supersede previous ones but, instead, subscribe for previous event list plus the one in the current call. All subscription done with this directive will be received, within the application, via a single listener.
Apply to	ApplicationManager Class All acquired virtual components of class "CUSSCntl"
Available to	AL application in INITIALIZE or ACTIVE state Service Provider System Manager Application Provider System Manager
Access	Shared, local/remote, synchronous

Structure sent	Structure of {
	Application Token: defined in Section 3.1.4
	Action: set of {SUBSCRIBE, DISCARD} (subscribe to receive the event, discard to not receive the event)
	Event list selection: structure defined in 3.1.12 (specifies the events to register)
	Listener: reference (object reference of the listener)
	Correlation: defined in Section 3.1.5 (user data that is submitted with each event send to the listener)
	}

Code returned	**Function Return Code:** defined in Appendix A
Structure returned	**Event**: structure defined in Section 3.1.11

3.3.6 waitEvent

Description	Application is waiting for an event to occur. To wait for an event, the application must have subscribed to it via the **acquire** (Section 3.6.1) or registerEvent (Section 3.3.5**)** directives. The directive will be completed at event occurrence (any or all in the list) or as timeout expired.
Apply to	ApplicationManager Class All acquired virtual components of class "CUSSCntl"
Available to	AL application
Access	Shared, local/remote, synchronous

Structure sent	Structure of {
	Timeout[9]: defined in Section 3.1.3
	Application Token: defined in Section 3.1.4
	Event list selection: structure defined in Section 3.1.12
	}

Code returned	**Function Return Code:** defined in Appendix A

[9] Positive and negative timeout values have the same effect for `waitEvent` directive

Structure returned	**Event**: structure defined in Section 3.1.11

3.4 Application Manager Interface (AMI) Directives[10]

The Application Manager interface is accessed by the AL application using CORBALOC as follows: **`corbaloc:<kiosk-IP address>:20000/ApplicationManager or corbaloc:<kiosk-host name>:20000/ApplicationManager,`** provided that the kiosk host name could be resolved to a valid IP address.
The AMI directives constitute of all the Management Interface directives, defined in Section 3.3, and the following two directives: initRequest and notify.

3.4.1 initRequest

Description	The application now wants to initialize/re-initialize. This is a blocking call. After this directive returns the application is allowed to initialize. This handling ensures that initialization is serialized for all applications. This is necessary because applications may load e.g. PECTAB to an ATB printer, which can be done by only one application at a time.
Apply to	ApplicationManager Class
Available to	Airline application in STOPPED state after being loaded by application manager
Access	Shared, local/remote, synchronous
Structure sent	**Application Token:** defined in Section 3.1.4
Code returned	**Function Return Code:** defined in Appendix A
Structure returned	**Event**: structure defined in Section 3.1.11

3.4.2 notify

Description	This directive is used by the application to request a state change from CUSS Application Manager, which will change the application state if request is approved.
Apply to	ApplicationManager Class
Available to	AL Application in a neighborhood state
Access	Shared, local/remote, synchronous

Structure sent	Structure of {
	Application Token: defined in Section 3.1.4**,** token of the requesting application
	Kiosk-Application ID[11]: defined in Section 3.1.10, Kiosk Application whose state to be changed.
	State Transition: number (event code {101 to 130} defined in Appendix A)
	}

Code returned	**Function Return Code:** defined in Appendix A
Structure returned	**Event**: structure defined in Section 3.1.11

[10] AMI directives could be made available to CLA if the platform provider chooses to base the communication between CLA and CAM based on the AMI interface. As CLA is an integral part of the CUSS platform, this is an internal decision to be made by platform provider.
[11] This maybe useful for CLA to inform CAM which kiosk application is activated. This depends whether CLA use this interface to communicate to CAM. This is a design decision left to the platform provider.

3.5 System Manager Interface (SMI) Directives

The system manager interface is accessed by SP/AL system managers using CORBALOC as follows: **`corbaloc:<kiosk-IP address>: 20001/ServiceProviderInterface`** or **`corbaloc:<kiosk-host name>:20001/ServiceProviderInterface`,** provided that the kiosk host name could be resolved to a valid IP address.
SMI directives are comprised of all the Management Interface directives defined earlier in Section 3.3, as well as the following directives described below: **`load, resume, resumeAll, stop, stopAll, suspend,`** and **`suspendAll`**. These directives will result in one or more events generated to the applicable application(s) to inform them of their state change. Refer to Section 3.7.3 for more information about these events.
Note: The AL system Manager can affect only application(s) of the same AL.

3.5.1 load

Description	Ask CUSS application Manager to load an application (realize Load state transition).
Apply to	AL Application in DISABLED state (only available for SP System Manager upon human intervention) or in STOPPED state
Available to	Service Provider System Manager Application Provider System Manager
Access	Shared, local/remote, synchronous/asynchronous
Structure sent	**Timeout:** Timeout value for the call; defined in Section 3.1.3 **Application Token:** defined in Section 3.1.4, token of the requesting SM application **Kiosk-Application ID:** defined in Section 3.1.10, ID of Kiosk Application to be loaded.
Code returned	**Function Return Code:** defined in Appendix A
Structure returned	**Event**: structure defined in Section 3.1.11

Note: If there is another ACTIVE application running, **`load`** should queue the request until the application completes its session. In case of synchronous call, **`load`** will block until its timeout elapses or it is allowed to start executing.

A system manager is not allowed to load an application that was stopped by the other system manager.

3.5.2 resume

Description	Resume a suspended application to its previous state (AVAILABLE, UNAVAILABLE or SUSPENDED).
Apply to	AL application that was put in SUSPENDED state by the same system manager
Available to	Service Provider System Manager Application Provider System Manager
Access	Shared, local/remote, synchronous
Structure sent	**Application Token:** defined in Section 3.1.4, token of the requesting SM application **Kiosk-Application ID:** defined in Section 3.1.10, ID of Kiosk Application to be resumed.
Code returned	**Function Return Code:** defined in Appendix A
Structure returned	**Event**: structure defined in Section 3.1.11

Note: If an application is suspended by its own AL System Manager and the SP System Manager, it is required that both system managers resume the application to get back to its pre-suspended state.

3.5.3 resumeAll

Description	Resume all suspended applications to their previous states (AVAILABLE, UNAVAILABLE, or SUSPENDED).
Apply to	AL applications that were put in SUSPENDED state by the same system manager
Available to	Service Provider System Manager Application Provider System Manager
Access	Shared, local/remote, synchronous
Structure sent	**Application Token:** defined in Section 3.1.4, token of the requesting SM application
Code returned	**Function Return Code:** defined in Appendix
Structure returned	**Event**: structure defined in Section 3.1.11

3.5.4 stop

Description	Stop (i.e. unload) an application (realize Stop state transition)
Apply to	AL application in INITIALIZE, UNAVAILABLE, AVAILABLE or ACTIVE state or that was put into SUSPENDED state by the same system manager
Available to	Service Provider System Manager Application Provider System Manager
Access	Shared, local/remote, synchronous
Structure sent	**Application Token:** defined in Section 3.1.4, token of the requesting SM application. **Kiosk-Application ID:** defined in Section 3.1.10, ID of Kiosk Application to be stopped.
Code returned	**Function Return Code:** defined in Appendix A
Structure returned	**Event**: structure defined in Section 3.1.11

3.5.5 stopAll

Description	Stop all applications (realize Stop state transition).
Apply to	AL applications in INITIALIZE, UNAVAILABLE, AVAILABLE or ACTIVE state AL applications that was put in SUSPENDED state by the same system manager.
Available to	Service Provider System Manager Application Provider System Manager
Access	Shared, local/remote, synchronous
Structure sent	**Application Token:** defined in Section 3.1.4, token of the requesting SM application
Code returned	**Function Return Code:** defined in Appendix A
Structure returned	**Event**: structure defined in Section 3.1.11

3.5.6 suspend

Description	Suspend an application.
Apply to	AL applications in AVAILABLE, UNAVAILABLE or SUSPENDED state
Available to	Service Provider System Manager Application Provider System Manager

	(only the same SM can issue a resume later)
Access	Shared, local/remote, synchronous
Structure sent	**Application Token:** defined in Section 3.1.4, token of the requesting SM application **Kiosk-Application ID:** defined in Section 3.1.10, ID of Kiosk Application to be suspended.
Code returned	**Function Return Code:** defined in Appendix A
Structure returned	**Event**: structure defined in Section 3.1.11

3.5.7 suspendAll

Description	Suspend all applications.
Apply to	AL applications in AVAILABLE, UNAVAILABLE or SUSPENDED state
Available to	Service Provider System Manager Application Provider System Manager (only the same SM can issue a **`resume`** or **`resumeAll`** later)
Access	Shared, local/remote, synchronous
Structure sent	**Application Token:** defined in Section 3.1.4, token of the requesting SM application
Code returned	**Function Return Code:** defined in Appendix A
Structure returned	**Event**: structure defined in Section 3.1.11

Note: If one of the applicable applications is in ACTIVE state, like **`suspend`**, **`suspendAll`** will fail and return RC_STATE as function return code.

3.6 Device Component Interface (DCI) Directives

DCI directives manage virtual components controlled by CUSS. They are divided into four categories: component directives, data directives, document directives and event directives.
Component directives are: acquire, disable, enable, query, release, setup and test.
Data directives are: receive and send.
Document directives are: offer and retain.
Event directive is: cancel.

Please read Section 3.8 below for some examples of how certain events and status codes correspond to real device behaviour for MediaInput devices.

3.6.1 acquire

Description	Make the virtual component available for an application. The application could at the same time subscribe to a component specific listener associate to the component acquired.
Apply to	All released virtual components of class “Peripheral” or “NativeDevice”. The initial state of these virtual components is RELEASED.
Available to	AL application in INITIALIZE, UNAVAILABLE, AVAILABLE or ACTIVE state Service Provider System Manager Application Provider System Manager
Access	Shared, local/remote, synchronous/asynchronous

Structure sent	Structure of {
	Timeout: Timeout value for the call; defined in Section 3.1.3
	Application Token: defined in Section 3.1.4
	Event list selection: structure defined in 3.1.12 (specifies the component events to register i.e. event filtering)
	Listener: reference (object reference of the listener)
	Correlation: defined in Section 3.1.5 (user data that is submitted with each event send to the listener)
	}

Code returned	**Function Return Code:** defined in Appendix A
Structure returned	**Event**: structure defined in Section 3.1.11

The status code in the retuned event represents the device status and depends on the virtual component this directive is applied to as shown in the following table:

Function: acquire	**Virtual Component Types**											
Status Code	Capture	DataInput	DataOutput	UserInput	UserOutput	Dispenser	Feeder	MediaInput	MediaOutput	Display	Network	Storage
OK	X	X	X	X	X	X	X	X	X	X	X	X
TIMEOUT	X	X	X	X	X	X	X	X	X	X	X	X

Function: acquire	Virtual Component Types											
Status Code	Capture	DataInput	DataOutput	UserInput	UserOutput	Dispenser	Feeder	MediaInput	MediaOutput	Display	Network	Storage
WRONG_STATE[12]	X	X	X	X	X	X	X	X	X	X	X	X
CANCELLED	X	X	X	X	X	X	X	X	X	X	X	X
SOFTWARE_ERROR	X	X	X	X	X	X	X	X	X	X	X	X
MEDIA_JAMMED	X					X	X	X	X			
MEDIA_MISPLACED						X	X	X	X			
MEDIA_PRESENT						X		X	X			
MEDIA_ABSENT	X					X		X	X			
MEDIA_HIGH	X					X						X
MEDIA_FULL	X					X						X
MEDIA_LOW							X					
MEDIA_EMPTY							X					
MEDIA_DAMAGED												X
MEDIA_INCOMPLETELY_INSERTED								X				
CONSUMABLES									X			
HARDWARE_ERROR	X	X	X	X	X	X	X	X	X	X	X	X
CRITICAL_SOFTWARE_ERROR	X	X	X	X	X	X	X	X	X	X	X	X
NOT_REACHABLE	X	X	X	X	X	X	X	X	X	X	X	X
NOT_RESPONDING	X	X	X	X	X	X	X	X	X	X	X	X
THRESHOLD_ERROR	X	X	X	X	X	X	X	X	X	X	X	X
THRESHOLD_USAGE	X	X	X	X	X	X	X	X	X	X	X	X
CONFIGURATION_ERROR		X	X	X	X			X	X	X	X	X

3.6.2 disable

Description	Make the virtual component unavailable for the user (e.g. disable a reader device from a document insertion).
Apply to	All acquired and enabled virtual components of class “User” excluding “NativeDevice” components .
Available to	AL Application in ACTIVE state Service Provider System Manager
Access	Exclusive, local/remote, synchronous/asynchronous
Structure sent	**Timeout:** Timeout value for the call; defined in Section 3.1.3 **Application Token:** defined in Section 3.1.4
Code returned	**Function Return Code:** defined in Appendix A
Structure returned	**Event**: structure defined in Section 3.1.11

[12] WRONG_STATE is used in case of double acquire calls.

Note 1 (taken from CUSS 1.0 Addendum A.1.12):
The platform will maintain in the enabled state any component upon which an application calls enable(), until that application calls disable(), no matter how many documents are processed.

In some cases, a component is disabled practically by its physical limitations. For example, a motorized card reader cannot read additional cards, even while logically enabled by the application, until that card is offered and removed, or retained. If for this or any other reason the physical component becomes disabled after reading a document (for example because of its firmware logic) then the platform must automatically re-enable it (unless the application explicitly disables the component directly immediately after a read.)

The status code in the retuned event represents the function call status and depends on the virtual component this directive is applied to as shown in the following table:

Function: disable	**Virtual Component Types**				
Status Code	UserInput	UserOutput	Dispenser	MediaInput	MediaOutput
OK	X	X	X	X	X
TIMEOUT	X	X	X	X	X
WRONG_STATE	X	X	X	X	X
CANCELLED	X	X	X	X	X
SOFTWARE_ERROR	X	X	X	X	X
OUT_OF_SEQUENCE[13]	X	X	X	X	X
DATA_PRESENT	X	X			
CONSUMABLES					X
HARDWARE_ERROR	X	X	X	X	X
CRITICAL_SOFTWARE_ERROR	X	X	X	X	X
NOT_REACHABLE	X	X	X	X	X
NOT_RESPONDING	X	X	X	X	X
THRESHOLD_ERROR	X	X	X	X	X
THRESHOLD_USAGE	X	X	X	X	X
CONFIGURATION_ERROR	X	X	X	X	X

3.6.3 enable

Description	Make the virtual component available for the user (e.g. enable a reader device for a document insertion).
Apply to	All acquired and disabled virtual components of class “User” excluding “NativeDevice”

[13] OUT_OF_SEQUENCE is used in case of double disable calls.

	components. By default, all virtual components are disabled when they are acquired.
Available to	AL Application in ACTIVE state Service Provider System Manager
Access	Exclusive, local/remote, synchronous/asynchronous
Structure sent	**Timeout:** Timeout value for the call; defined in Section 3.1.3 **Application Token:** defined in Section 3.1.4
Code returned	**Function Return Code: defined in** Appendix A
Structure returned	**Event**: structure defined in Section 3.1.11

Note 1 (taken from CUSS 1.0 Addendum A.1.12):
The platform will maintain in the enabled state any component upon which an application calls enable(), until that application calls disable(), no matter how many documents are processed.

In some cases, a component is disabled practically by its physical limitations. For example, a motorized card reader cannot read additional cards, even while logically enabled by the application, until that card is offered and removed, or retained. If for this or any other reason the physical component becomes disabled after reading a document (for example because of its firmware logic) then the platform must automatically re-enable it (unless the application explicitly disables the component directly immediately after a read.)

The status code in the retuned event represents the device status and depends on the virtual component this directive is applied to as shown in the following table:

Function: Enable	**Virtual Component Types**				
Status Code	UserInput	UserOutput	Dispenser	MediaInput	MediaOutput
OK	X	X	X	X	X
TIMEOUT	X	X	X	X	X
WRONG_STATE	X	X	X	X	X
CANCELLED	X	X	X	X	X
SOFTWARE_ERROR	X	X	X	X	X
OUT_OF_SEQUENCE[14]	X	X	X	X	X
MEDIA_JAMMED			X	X	X
MEDIA_MISPLACED			X	X	X
MEDIA_PRESENT			X	X	X
MEDIA_ABSENT			X	X	X
MEDIA_HIGH			X		
MEDIA_FULL			X		
MEDIA_INCOMPLETELY_INSERTED				X	

[14] OUT_OF_SEQUENCE is used in case of double enable calls.

DATA_PRESENT	X	X		X	
CONSUMABLES					X
HARDWARE_ERROR	X	X	X	X	X
CRITICAL_SOFTWARE_ERROR	X	X	X	X	X
NOT_REACHABLE	X	X	X	X	X
NOT_RESPONDING	X	X	X	X	X
THRESHOLD_ERROR	X	X	X	X	X
THRESHOLD_USAGE	X	X	X	X	X
CONFIGURATION_ERROR	X	X		X	X

3.6.4 query

Description	Return the state/status of the virtual component.
Apply to	All acquired virtual components of class “Peripheral” and “ NativeDevice” and virtual components of class “ApplicationComponent”
Available to	AL Application in INITIALIZE, UNAVAILABLE, AVAILABLE or ACTIVE state Service Provider System Manager Application Provider System Manager
Access	Shared, local/remote, synchronous/asynchronous
Structure sent	**Timeout:** Timeout value for the call; defined in Section 3.1.3 **Application Token:** defined in Section 3.1.4
Code returned	**Function Return Code:** defined in Appendix A
Structure returned	**Event**: structure defined in Section 3.1.11

The event code in the returned event should contain the device component state or the application state (if **query** is applied on an *Application* component).
The status code in the retuned event represents the device status (or the last known device status if the device component is busy) and depends on the virtual component this directive is applied to as shown in the following table:

Note 1:
A Dispenser component will return OK if it does not have any ability to detect if a document is present (such as the case of a very simple paper path without any sensors.)

Function: query	**Virtual Component Types**											
Status Code	Capture	DataInput	DataOutput	UserInput	UserOutput	Dispenser	Feeder	MediaInput	MediaOutput	Display	Network	Storage
OK	X	X	X	X	X	X	X	X	X	X	X	X
TIMEOUT	X	X	X	X	X	X	X	X	X	X	X	X
WRONG_STATE	X	X	X	X	X	X	X	X	X	X	X	X
CANCELLED	X	X	X	X	X	X	X	X	X	X	X	X
SOFTWARE_ERROR	X	X	X	X	X	X	X	X	X	X	X	X

Function: query	Virtual Component Types											
Status Code	Capture	DataInput	DataOutput	UserInput	UserOutput	Dispenser	Feeder	MediaInput	MediaOutput	Display	Network	Storage
MEDIA_JAMMED	X					X	X	X	X			
MEDIA_MISPLACED						X	X	X	X			
MEDIA_PRESENT						X		X	X			
MEDIA_ABSENT	X					X		X	X			
MEDIA_HIGH	X					X						X
MEDIA_FULL	X					X						X
MEDIA_LOW							X					
MEDIA_EMPTY							X					
MEDIA_DAMAGED												X
MEDIA_INCOMPLETELY_INSERTED								X				
DATA_PRESENT		X		X				X				
CONSUMABLES									X			
HARDWARE_ERROR	X	X	X	X	X	X	X	X	X	X	X	X
CRITICAL_SOFTWARE_ERROR	X	X	X	X	X	X	X	X	X	X	X	X
NOT_REACHABLE	X	X	X	X	X	X	X	X	X	X	X	X
NOT_RESPONDING	X	X	X	X	X	X	X	X	X	X	X	X
THRESHOLD_ERROR	X	X	X	X	X	X	X	X	X	X	X	X
THRESHOLD_USAGE	X	X	X	X	X	X	X	X	X	X	X	X
CONFIGURATION_ERROR		X	X	X	X			X	X	X	X	X

3.6.5 release

Description	Make the virtual component unavailable for an application and unsubscribe the event listener relative to the component. All pending asynchronous directives will be automatically cancelled if not cancelled by the application itself.
Apply to	All acquired virtual components of class “Peripheral” or “NativeDevice”.
Available to	AL Application in INITIALIZE, UNAVAILABLE, AVAILABLE, or ACTIVE state Service Provider System Manager Application Provider System Manager
Access	Shared, local/remote, synchronous
Structure sent	**Timeout:** Timeout value for the call; defined in Section 3.1.3 **Application Token:** defined in Section 3.1.4
Code returned	**Function Return Code:** defined in Appendix A
Structure returned	**Event**: structure defined in Section 3.1.11

The status code in the retuned event represents the function call status and depends on the virtual component this directive is applied to as shown in the following table:

Function: release	**Virtual Component Types**											
Status Code	Capture	DataInput	DataOutput	UserInput	UserOutput	Dispenser	Feeder	MediaInput	MediaOutput	Display	Network	Storage
OK	X	X	X	X	X	X	X	X	X	X	X	X
TIMEOUT	X	X	X	X	X	X	X	X	X	X	X	X
WRONG_STATE[15]	X	X	X	X	X	X	X	X	X	X	X	X
SOFTWARE_ERROR	X	X	X	X	X	X	X	X	X	X	X	X

3.6.6 setup

Description	Set the virtual component and set up its profile for the application
Apply to	All acquired virtual components of class “Peripheral” excluding “NativeDevice” components.
Available to	Service Provider System Manager (only if kiosk is not in use) AL application in INITIALIZE or ACTIVE state
Access	Exclusive, local/remote, synchronous/asynchronous
Structure sent	Structure of { **Timeout:** Timeout value for the call; defined in Section 3.1.3 **Application Token:** defined in Section 3.1.4 **Data**: structure defined in Section 3.1.9[16]

[15] WRONG_STATE is used in case of double acquire calls.

[16] Data is used to download new “resources” to the component (e.g. PECTAB & logo download). If component does not support this data type, the call will be rejected and return RC_PARAMETER as function return code.

	}
Code returned	**Function Return Code:** defined in Appendix A
Structure returned	**Event**: structure defined in Section 3.1.11

Note: When the application becomes ACTIVE, the platform ensures to select the proper context of that application set by commands used in `setup` directive.

Only the following AEA commands are acceptable in case the data input parameter is *aeaDataType*: **CT, PT, PC, PS, LT, LC, LS, FT, FC, FS, FA, FR, TT, TC, TA, AV[17], ZS[18], PV, RI, RC, ES and EP**. For LT, LC, LS, logos shall be in PCX format (See Appendix D.)

BT command with no parameters is allowed. Any parameter sent with BT (trying to setup the bin) will be ignored. Any other commands must result in RC_UNAUTHORIZED. For more information on using the AEA standard in CUSS, please see Appendix D.
Bag tag printers shall support the BTT request.

When the application becomes ACTIVE, the platform ensures to select the proper context of that application set by commands used in `setup` directive.
The status code in the retuned event depends on the virtual component this directive is applied to as shown in the following table:

Function: setup	**Virtual Component Types**								
Status Code	Capture	DataInput	DataOutput	UserInput	UserOutput	Dispenser	Feeder	MediaInput	MediaOutput
OK	X	X	X	X	X	X	X	X	X
TIMEOUT	X	X	X	X	X	X	X	X	X
WRONG_STATE	X	X	X	X	X	X	X	X	X
CANCELLED	X	X	X	X	X	X	X	X	X
SOFTWARE_ERROR	X	X	X	X	X	X	X	X	X
OUT_OF_SEQUENCE				X	X			X	X
FORMAT_ERROR	X	X	X	X	X	X	X	X	X
LENGTH_ERROR		X	X	X	X			X	X
DATA_MISSING		X	X	X	X			X	X
CONSUMABLES									X
HARDWARE_ERROR	X	X	X	X	X	X	X	X	X
CRITICAL_SOFTWARE_ERROR	X	X	X	X	X	X	X	X	X
NOT_REACHABLE	X	X	X	X	X	X	X	X	X
NOT_RESPONDING	X	X	X	X	X	X	X	X	X
THRESHOLD_ERROR	X	X	X	X	X	X	X	X	X

[17] AV is used to query the revision of the AEA standard supported by the printer
[18] ZS is used to determine of a bagtag printer can print in color.

THRESHOLD_USAGE	X	X	X	X	X	X	X	X	X
CONFIGURATION_ERROR		X	X	X	X			X	X

3.6.7 test

Description	Test the virtual component and the real component as deep as possible. If the component is a physical device then the device driver should be accessed but the physical device should not be exercised.
Apply to	All acquired virtual components of class “Peripheral” excluding “NativeDevice” components.
Available to	Service Provider System Manager (only if kiosk is not in use)
Access	Exclusive, local/remote, synchronous/asynchronous.
Structure sent	**Timeout:** Timeout value for the call; defined in Section 3.1.3 **Application Token:** defined in Section 3.1.4
Code returned	**Function Return Code:** defined in Appendix A
Structure returned	**Event**: structure defined in Section 3.1.11

The status code in the retuned event represents the device status and depends on the virtual component this directive is applied to as shown in the following table:

Function: test	**Virtual Component Types**								
Status Code	Capture	DataInput	DataOutput	UserInput	UserOutput	Dispenser	Feeder	MediaInput	MediaOutput
OK	X	X	X	X	X	X	X	X	X
TIMEOUT	X	X	X	X	X	X	X	X	X
WRONG_STATE	X	X	X	X	X	X	X	X	X
CANCELLED	X	X	X	X	X	X	X	X	X
SOFTWARE_ERROR	X	X	X	X	X	X	X	X	X
OUT_OF_SEQUENCE	X			X	X	X		X	X
MEDIA_JAMMED	X					X	X	X	X
MEDIA_MISPLACED						X	X	X	X
MEDIA_PRESENT						X		X	X
MEDIA_ABSENT	X					X		X	X
MEDIA_HIGH	X					X			
MEDIA_FULL	X					X			
MEDIA_LOW							X		
MEDIA_EMPTY							X		
MEDIA_INCOMPLETELY_INSERTED								X	
DATA_PRESENT		X		X				X	
CONSUMABLES									X
HARDWARE_ERROR	X	X	X	X	X	X	X	X	X
CRITICAL_SOFTWARE_ERROR	X	X	X	X	X	X	X	X	X

NOT_REACHABLE	X	X	X	X	X	X	X	X	X
NOT_RESPONDING	X	X	X	X	X	X	X	X	X
THRESHOLD_ERROR	X	X	X	X	X	X	X	X	X
THRESHOLD_USAGE	X	X	X	X	X	X	X	X	X
CONFIGURATION_ERROR		X	X	X	X			X	X

3.6.8 Data Directives

The two data directives listed below, namely **`receive`** and **`send`**, are associated to data handling. They are both are implemented with synchronous and asynchronous interface calls available to the application.
All messages received by an application within an event resulting of a directive execution must be of the same type (format) as the one used by the application when the directive was issued.

3.6.8.1 receive

Description	Make the data from the virtual component available to the application. The application has to call this directive to get unsolicited data from a virtual component.
Apply to	All acquired virtual components of class “Input” excluding “NativeDevice” components. If the component is also of class "User", it has to be enabled before.
Available to	AL Application in ACTIVE state Service Provider System Manager
Access	Exclusive, local/remote, synchronous/asynchronous
Structure sent	**Timeout:** Timeout value for the call; defined in Section 3.1.3 **Application Token:** defined in Section 3.1.4
Code returned	**Function Return Code:** defined in Appendix A
Structure returned	**Event**: structure defined in Section 3.1.11

Note:
It is the responsibility of the platform to ensure that the device component is ready to receive data (e.g. Shutter is open in ATB) assuming the application has previously enabled the device component.

Note 1:
Some data obtained from the platform via the receive() directive may be considered sensitive data, such as payment card raw track information. Please review Section 1.7: Data Security Considerations for important information on how applications should handle, process, and forward sensitive information that is provided by the CUSS platform.

Note 2 (CUSS 1.0 Addendum A.1.18):
The platform will not include any device-specific separators (such as track separator or sentinel characters), or filler/error characters, and will only return the standard data. If the physical reader substitutes characters, for example for unreadable OCR character positions, the data record status will indicate that it is a bad read but return as much information as possible.

If the media being read has a logical multi-track arrangement, then each track is returned as a separate msgData data “track”. Examples of this include 2 or 3-track magnetic cards, 2-track

standard passport MRZ data, 3-track National ID Card OCR data, etc. (Any valid multi-track document data can be received in this fashion. CUSS is not limited to only certain types of documents.)

Note 3 (CUSS 1.0 Addendum A.1.24):
Application developers should be aware that any MediaInput component (passport readers, card readers, etc) might be linked to a Dispenser component. If this is the case, then the CUSS application must call offer() on the linked Dispenser component in order to return the document to the user. If the application does not handle this case, it is possible that document (card or passport, etc) will be captured by the platform and not returned to the user.

Note 4 (based on CUSS 1.0 Addendum A.1.41):
An application can call receive() only once to get data from a Input component after that kiosk device has obtained data from the user (card swipe, etc.) The platform must not "cache" Input data (DataInput, MediaInput, UserInput, Conveyor) data after the initial receive() by the application.

For example, in a motorized card reader, if the application obtains the card data via receive() and does not call offer() to eject the card, subsequent calls to receive() will not return the data again (DATA_MISSING.) Only if the card is ejected, and if the customer again inserts a card, will receive() again provide data – from the new card.

For example, in a swipe reader, the application calls enable() and then receive() with a long timeout. That receive() times out or returns a DATA_PRESENT event with the card data. If the application calls receive() again, because the device is still enabled that call will block until a new card is swiped.

For example, an application calls enable() and then waits for DATA_PRESENT on its event listener, then calls receive() to obtain the data. Another call to receive() would not return the data again (it would either give DATA_MISSING for a motorized reader if no offer() had been called, or TIMEOUT if another card wasn't swiped or inserted within the specified timeout.)

Note 5 (CUSS FOID Addendum):
If the application calls receive() on a card reader after the customer has read a payment card, the platform will return truncated track data. Please review Chapter 8 for more information.

Note 6 (multiple simultaneous read tracks):

Some devices such as barcode scanners, may also return multiple "tracks" if the reader is capable of detecting multiple barcodes on the same document or item – for example multiple bag tags or barcodes on a bag. If an application wishes to support this type of device, it may need additional logic to detect and consume any additional tracks of data provided by the platform.

The status code in the retuned event represents the function call status and depends on the virtual component this directive is applied to as shown in the following table:

Function: receive	**Virtual Component Types**		
Status Code	DataInput	UserInput	MediaInput
OK	X	X	X
TIMEOUT	X	X	X
WRONG_STATE	X	X	X
CANCELLED	X	X	X
SOFTWARE_ERROR	X	X	X
OUT_OF_SEQUENCE	X	X	X
FORMAT_ERROR	X	X	X
LENGTH_ERROR	X	X	X
DATA_MISSING	X	X	X
HARDWARE_ERROR	X	X	X
CRITICAL_SOFTWARE_ERROR	X	X	X
NOT_REACHABLE	X	X	X
NOT_RESPONDING	X	X	X
THRESHOLD_ERROR	X	X	X
THRESHOLD_USAGE	X	X	X
CONFIGURATION_ERROR	X	X	X

3.6.8.2 send

Description	Send data from the application to the virtual component.
Apply to	All acquired virtual component of class “Output” excluding “NativeDevice” components. If the component is also of class "User", it has to be enabled before.
Available to	AL Application in ACTIVE state Service Provider System Manager
Access	Exclusive, local/remote, synchronous/asynchronous

Structure sent	Structure of {
	Timeout: Timeout value for the call; defined in Section 3.1.3
	Application Token: defined in Section 3.1.4
	Data: structure defined in Section 3.1.9
	}

Code returned	**Function Return Code:** defined in Appendix A
Structure returned	**Event**: structure defined in Section 3.1.11

Note: The data stream in the returned event (if any) should be the same type of the input data parameter. For example, if an application has sent a AEA message, it should expect an AEA message back in the data field of the returned event.

Only the following AEA commands are acceptable if the input data parameter is *aeaDataType*: **CP, CI, TK, TI, TR**. Any other commands must result in RC_UNAUTHORIZED.

Bag tag printers shall also support the BTP request.

Other AEA commands (e.g. to operate insertion, eject, escrow, etc.) are implemented by specific CUSS directives like `setIOMode, offer, retain,` etc.
VSR (Void Stacker Ribbon indicator) field is mandatory in the ATB responses.
The status code in the retuned event represents the function call status and depends on the virtual component this directive is applied to as shown in the following table:

Note 1:
Please see Section 3.2.3 for the expected behavior when a component linked to the MediaOutput component prevents the send() from completing.

Function: send	**Virtual Component Types**		
Status Code	DataOutput	UserOutput	MediaOutput
OK	X	X	X
TIMEOUT	X	X	X
WRONG_STATE	X	X	X
CANCELLED	X	X	X
SOFTWARE_ERROR	X	X	X
OUT_OF_SEQUENCE		X	X
FORMAT_ERROR	X	X	X
LENGTH_ERROR	X	X	X
DATA_MISSING	X	X	X
CONSUMABLES			X
HARDWARE_ERROR	X	X	X
CRITICAL_SOFTWARE_ERROR	X	X	X
NOT_REACHABLE	X	X	X
NOT_RESPONDING	X	X	X
THRESHOLD_ERROR	X	X	X
THRESHOLD_USAGE	X	X	X

Function: send	Virtual Component Types		
Status Code	DataOutput	UserOutput	MediaOutput
CONFIGURATION_ERROR	X	X	X

3.6.9 Document Directives

The two document directives listed below, namely **`offer`** and **`retain`**, allow document manipulation. They are both implemented with synchronous and asynchronous interface calls available to the application.

3.6.9.1 offer

Description	Offer the document from the virtual component to the user or to an other component
Apply to	All acquired virtual component of class " Feeder" or "Dispenser".
Available to	AL application in ACTIVE state Service Provider System Manager
Access	Exclusive, local/remote, synchronous/asynchronous
Structure sent	**Timeout:** Timeout value for the call; defined in Section 3.1.3 **Application Token:** defined in Section 3.1.4
Code returned	**Function Return Code:** defined in Appendix A
Structure returned	**Event**: structure defined in Section 3.1.11

Note: Directive `offer` is only required in case of manual feeder or real dispenser. Some examples are: a MediaOutput (e.g. Card writer) that requires an explicit form feed or an ESCROW device.

The status code in the retuned event represents the device status and depends on the virtual component this directive is applied to as shown in the following table:

Function: Offer	Virtual Component Types	
Status Code	Dispenser	Feeder
OK	X	X
TIMEOUT	X	X
WRONG_STATE[19]	X	X

[19] Calling offer when Feeder is empty (so in UNAVAILABLE state) results in status code WRONG_STATE.

CANCELLED	X	X
SOFTWARE_ERROR	X	X
OUT_OF_SEQUENCE	X	
MEDIA_JAMMED	X	X
MEDIA_MISPLACED	X	X
MEDIA_PRESENT	X	
MEDIA_ABSENT	X	
MEDIA_HIGH	X	
MEDIA_FULL	X	
MEDIA_LOW		X
MEDIA_EMPTY[20]		X
HARDWARE_ERROR	X	X
CRITICAL_SOFTWARE_ERROR	X	X
NOT_REACHABLE	X	X
NOT_RESPONDING	X	X
THRESHOLD_ERROR	X	X
THRESHOLD_USAGE	X	X

[20] If the feeder has one document left, calling offer will be successful with status code MEDIA_EMPTY (as the new device status).

Note 2 (CUSS 1.0 Addendum A.1.11):

If a Dispenser component is real then an offer() directive is required to make the document(s) available to the user (such as ejecting a card from a motorized reader, or opening an escrow door).

If a Dispenser component is virtual, documents are available to the user immediately even without a call to offer().

If the Dispenser component (real or virtual) can detect when documents have been removed by the user, the offer() directive is a **blocking** call that returns in normal conditions only after the documents are taken (or the request times out.) A virtual dispenser can be blocking, such as is the case for a paper output shoot with document sensor, which is immediately available to the user but can detect when documents are present.

If a dispenser is not blocking, as defined above, then the offer() and query() directives shall return status code OK of no other error condition is present.

If the dispenser is virtual and there is no sensor, then when the application calls offer() asynchronously the CUSS platform must respond asynchronously with status code OK if no other error conditions are present so the application can determine that no physical sensor is present in the component.

3.6.9.2 retain

Description	Capture the document in the virtual component that is associated to secure bin.
Apply to	All acquired motorized virtual components of class “Capture”.
Available to	AL application in ACTIVE state Service Provider System Manager
Access	Exclusive, local/remote, synchronous/asynchronous
Structure sent	**Timeout:** Timeout value for the call; defined in Section 3.1.3 **Application Token:** defined in Section 3.1.4
Code returned	**Function Return Code:** defined in Appendix A
Structure returned	**Event**: structure defined in Section 3.1.11

The status code in the retuned event represents the device status and depends on the virtual component this directive is applied to as shown in the following table:

Function: retain	**Virtual Component**
Status Code	Capture
OK	X
TIMEOUT	X
WRONG_STATE	X
CANCELLED	X
SOFTWARE_ERROR	X
OUT_OF_SEQUENCE	X
MEDIA_JAMMED	X
MEDIA_ABSENT	X
MEDIA_HIGH	X
MEDIA_FULL	X
HARDWARE_ERROR	X
CRITICAL_SOFTWARE_ERROR	X
NOT_REACHABLE	X
NOT_RESPONDING	X
THRESHOLD_ERROR	X
THRESHOLD_USAGE	X

3.6.10 Event Directives

The following directive handles event-related functionality.

3.6.10.1 cancel

Description	Cancel all pending (previously called in asynchronous mode) directives related to the component at the time of this directive usage.
Apply to	All acquired virtual components of class “Peripheral”.
Available to	AL application in ACTIVE state Service Provider System Manager
Access	Exclusive, local/remote, synchronous
Structure sent	**Application Token:** defined in Section 3.1.4
Code returned	**Function Return Code:** defined in Appendix A
Structure returned	**Event**: structure defined in Section 3.1.11

The status code in the retuned event represents the function call status and depends on the virtual component this directive is applied to as shown in the following table:

Function: cancel	**Virtual Component Types**								
Status Code	Capture	DataInput	DataOutput	UserInput	UserOutput	Dispenser	Feeder	MediaInput	MediaOutput
OK	X	X	X	X	X	X	X	X	X
TIMEOUT	X	X	X	X	X	X	X	X	X
WRONG_STATE	X	X	X	X	X	X	X	X	X
SOFTWARE_ERROR	X	X	X	X	X	X	X	X	X

3.6.11 Media High/Full for Dispenser Components

This section is taken from CUSS 1.0 Addendum A.1.38. It clarified the behaviour of Dispenser components which have a finite and practical limitation on the number of documents that can be held before being retrieved by the kiosk user. In CUSS, this situation typically applies to Printer devices.

When a print command is issued via the send() directive, the linked Dispenser component will return a status code of MEDIA_PRESENT, MEDIA_HIGH, or MEDIA_FULL if it can detect the presence of a document (via document sensor, etc) and OK if it does not have the ability to detect documents.

The platform must return MEDIA_FULL if it knows that no further documents can be printed (due to a physical printer or stacker/escrow limitation, for example.) CUSS applications should monitor the Dispenser status code for MEDIA_HIGH or MEDIA_FULL, and offer the documents in the Dispenser to the user whenever this status is reached (as well as when it has finished printing all its documents, if any remain.)

The status code MEDIA_FULL may either be a hard error of a soft error depending on the virtual component. A dispenser component that is full will have a status code MEDIA_FULL as a soft error. The component will remain available to offer the media to the user. The event will be a private event.

If the application attempts to print more documents when a linked Dispenser is full, that request will fail with a HARDWARE_ERROR (see Section 3.2.3 for more information.)

It is possible that a CUSS kiosk with a simple printer device is able to stack only one document. In this case, the platform will return MEDIA_FULL after each print request and the application must then wait for removal from the Dispenser prior to printing the next document.

The platform shall indicate via the Bin characteristics any practical capacity limitation of its Dispenser component (see Section 5.1.1 for more information.) Applications can then use this information to determine how many consecutive prints are possible.

3.7 Event Listener Interface (ELI)

An AL application MUST register an event listener (using **`registerEvent`** directive) in order that the application manager can communicate with the application via the callback directive of the event listener. In addition, an AL application may choose to listen (by registering its listener via the **`acquire`** directive) to events generated by device components. On the other hand, a system manager (SP/AL) may choose to register its event listener(s) if it is interested to receive events from the application manager and/or device components.
This section defines the event listener callback directive, and then lists the events generated by either the CUSS Application Manger or the device components. These events are either sent to all its listeners (AL application, SP/AL system manager, or a platform application) or returned to their callers (AL application, SP/AL system manager, or a platform application) in case of solicited events.

3.7.1 callback Directive

Description	Sends an event to a previously registered listener.
Apply to	AL Application Service Provider System Manager Application Provider System Manager
Available to	ApplicationManager Class All virtual components of class "Peripheral"
Access	local/remote, synchronous
Structure sent	**Event**: structure defined in Section 3.1.11
Structure returned	**None**

3.7.2 Device Components Events

The following is a list of all events generated by a device component and either sent to all its applicable listeners or returned to its caller in case of solicited events. The list is sorted by the event code, which represents either a state transition or the current state in case of no state transition, and includes all possible associated status codes. Refer to Sections 0 and 0 for full descriptions of the event codes and status codes.

Please read Section 3.8 below for some examples of how certain events and status codes correspond to real device behaviour for MediaInput devices.

Event Codes		
Event Code	**Description**	
001	**State transition OK & soft error = EVENTHANDLING_READY** Used for soft conditions and OK only.	
	Associated Status Codes	
	000	OK
	001	TIMEOUT
	002	WRONG_STATE
	003	CANCELLED
	004	SOFTWARE_ERROR
	006	OUT_OF_SEQUENCE
	102	MEDIA_MISPLACED
	103	MEDIA_PRESENT
	104	MEDIA_ABSENT
	105	MEDIA_HIGH
	107	MEDIA_LOW
	109	MEDIA_DAMAGED
	110	MEDIA_INCOMPLETELY_INSERTED
	201	FORMAT_ERROR
	202	LENGTH_ERROR
	203	DATA_MISSING
	205	DATA_PRESENT

Event Codes		
Event Code	**Description**	
002	**State transition Restart = UNAVAILABLE_RELEASED_PLATFORM** An authorized platform component has released a component in UNAVAILABLE state for any reason.	
	Associated Status Codes	
	801	CUSS_MANAGER_REQUEST
003	**State transition Hard error = EVENTHANDLING_UNAVAILABLE** Caused by a hard condition that made the component unusable.	
	Associated Status Codes	
	101	MEDIA_JAMMED
	102	MEDIA_MISPLACED
	106	MEDIA_FULL
	108	MEDIA_EMPTY
	301	CONSUMABLES
	302	HARDWARE_ERROR
	303	CRITICAL_SOFWARE_ERROR
	304	NOT_REACHABLE
	305	NOT_RESPONDING
	306	THRESHOLD_ERROR
	307	THRESHOLD_USAGE
	308	CONFIGURATION_ERROR
004	**State transition Release = UNAVAILABLE_RELEASED** An application has released a component in UNAVAILABLE state.	
	Associated Status Codes	
	000	OK
	004	SOFTWARE_ERROR
005	**State transition Release = READY_RELEASED_APPLICATION** An application has released a component in READY state.	
	Associated Status Codes	
	000	OK
	004	SOFTWARE_ERROR
006	**State change Restart = READY_RELEASED_PLATFORM** An authorized platform component has released a component in READY state for any reason.	
	Associated Status Codes	
	801	CUSS_MANAGER_REQUEST
007	**State transition Acquire = RELEASED_READY** An application has acquired a component that is working normally.	
	Associated Status Codes	
	000	OK
	004	SOFTWARE_ERROR

Event Codes		
Event Code	**Description**	
	102	MEDIA_MISPLACED
	103	MEDIA_PRESENT
	104	MEDIA_ABSENT
	105	MEDIA_HIGH
	107	MEDIA_LOW
	109	MEDIA_DAMAGED
	110	MEDIA_INCOMPLETELY_INSERTED
008	**State transition Acquire (hard error) = RELEASED_UNAVAILABLE** An application has acquired a component that is not working normally.	
	Associated Status Codes	
	101	MEDIA_ JAMMED
	102	MEDIA_MISPLACED
	103	MEDIA_PRESENT
	106	MEDIA_FULL
	107	MEDIA_ EMPTY
	301	CONSUMABLES
	302	HARDWARE_ERROR
	303	CRITICAL_SOFTWARE_ERROR
	304	NOT_REACHABLE
	305	NOT_RESPONDING
	306	THRESHOLD_ERROR
	307	THRESHOLD_USAGE
	308	CONFIGURATION_ERROR
Component States		
201	**State: RELEASED** No state transition. Component is in RELEASED state.	
	Associated Status Codes	
	000	OK
	001	TIMEOUT
	002	WRONG_STATE
	003	CANCELLED
	004	SOFTWARE_ERROR
202	**State: UNAVAILABLE** No state transition. Component is in UNAVAILABLE state.	
	Associated Status Codes	
	001	TIMEOUT
	002	WRONG_STATE
	003	CANCELLED
	004	SOFTWARE_ERROR
	101	MEDIA_ JAMMED
	102	MEDIA_MISPLACED
	103	MEDIA_PRESENT
	106	MEDIA_FULL
	107	MEDIA_ EMPTY
	301	CONSUMABLES

Event Codes		
Event Code	**Description**	
	302	HARDWARE_ERROR
	303	CRITICAL_SOFTWARE_ERROR
	304	NOT_REACHABLE
	305	NOT_RESPONDING
	306	THRESHOLD_ERROR
	307	THRESHOLD_USAGE
	308	CONFIGURATION_ERROR
203	**State: READY** No state transition. Component is in READY state.	
	Associated Status Codes	
	000	OK
	001	TIMEOUT
	002	WRONG_STATE
	003	CANCELLED
	004	SOFTWARE_ERROR
	006	OUT_OF_SEQUENCE
	102	MEDIA_MISPLACED
	103	MEDIA_PRESENT
	104	MEDIA_ABSENT
	105	MEDIA_HIGH
	107	MEDIA_LOW
	109	MEDIA_DAMAGED
	110	MEDIA_INCOMPLETELY_INSERTED
	201	FORMAT_ERROR
	202	LENGTH_ERROR
	203	DATA_MISSING
	205	DATA_PRESENT
210	**State: BUSY** Component is in BUSY transient state.	
	Associated Status Codes	
	000	OK
	102	MEDIA_MISPLACED
	103	MEDIA_PRESENT
	104	MEDIA_ABSENT
	105	MEDIA_HIGH
	107	MEDIA_LOW
	109	MEDIA_DAMAGED
	110	MEDIA_INCOMPLETELY_INSERTED
	201	FORMAT_ERROR
	202	LENGTH_ERROR
	205	DATA_PRESENT

3.7.3 CUSS Application Manager Events

The following is a list of all events[21] generated by CUSS Application Manager and either sent to all its applicable listeners or returned to its caller (AL application or SM) in case of solicited events. The list is sorted by the event code, which represents either a state transition or the current state in case of no state transition, and includes all possible associated status codes. Refer to Sections 0 and 0 for full descriptions of the event codes and status codes.

Event Codes		
Event Code	**Description**	
000	**State transition suspendAll, resumeAll, stopAll = EC_OK** Used in the returned event for calls to **suspendAll**, **resumeAll** or **stopAll** directives.	
	Associated Status Codes	
	000	OK
	001	TIMEOUT
	002	WRONG_STATE
	004	SOFTWARE_ERROR
	802	SP_SYSTEM_MANAGER_REQUEST
	803	AL_SYSTEM_MANAGER_REQUEST
Application State Transitions		
101	**State transition Disable = INITIALIZE_DISABLED** Requested by CUSS Application Manager.	
	Associated Status Codes	
	303	CRITICAL_SOFTWARE_ERROR
	305	NOT_RESPONDING
	306	THRESHOLD_ERROR
	801	CUSS_MANAGER_REQUEST
102	**State transition Disable = AVAILABLE_DISABLED** Requested by CUSS Application Manager.	
	Associated Status Codes	
	303	CRITICAL_SOFTWARE_ERROR
	305	NOT_RESPONDING
	306	THRESHOLD_ERROR
	801	CUSS_MANAGER_REQUEST
103	**State transition Disable = ACTIVE_DISABLED** Requested by CUSS Application Manager.	
	Associated Status Codes	
	303	CRITICAL_SOFTWARE_ERROR
	305	NOT_RESPONDING
	306	THRESHOLD_ERROR

[21] Event codes 117, 124, 125, 126, and 131 are no longer used in CUSS 1.0.

Event Codes		
Event Code	**Description**	
	310	KILL_TIMEOUT
	801	CUSS_MANAGER_REQUEST
104	**State transition Wait = UNAVAILABLE_AVAILABLE** Requested by an AL Application.	
	Associated Status Codes	
	805	AL_APPLICATION_REQUEST
105	**State transition Activate = AVAILABLE_ACTIVE** Requested by CUSS Application Manager upon information from Common Launch Application.	
	Associated Status Codes	
	804	CL_ APPLICATION_REQUEST
106	**State transition Wait = ACTIVE_AVAILABLE** Requested by AL Application.	
	Associated Status Codes	
	804	AL_APPLICATION_REQUEST
107[22]	**State transition Stop = INITIALIZE_STOPPED_STOP** Requested by AL Application, CUSS Application Manager or System Manager.	
	Associated Status Codes	
	000	OK
	001	TIMEOUT
	002	WRONG_STATE
	004	SOFTWARE_ERROR
	801	CUSS_MANAGER_REQUEST
	802	SP_SYSTEM_MANAGER_REQUEST
	803	AL_SYSTEM_MANAGER_REQUEST
	804	AL_APPLICATION_REQUEST
108[22]	**State transition Stop = AVAILABLE_STOPPED _STOP** Requested by AL Application, CUSS Application Manager or System Manager.	
	Associated Status Codes	
	000	OK
	001	TIMEOUT
	002	WRONG_STATE
	004	SOFTWARE_ERROR
	801	CUSS_MANAGER_REQUEST
	802	SP_SYSTEM_MANAGER_REQUEST
	803	AL_SYSTEM_MANAGER_REQUEST
	804	AL_APPLICATION_REQUEST

Event Codes		
Event Code	**Description**	
109[22]	**State transition Stop = ACTIVE_STOPPED _STOP** Requested by AL Application, CUSS Application Manager or System Manager.	
	Associated Status Codes	
	000	OK
	001	TIMEOUT
	002	WRONG_STATE
	004	SOFTWARE_ERROR
	801	CUSS_MANAGER_REQUEST
	802	SP_SYSTEM_MANAGER_REQUEST
	803	AL_SYSTEM_MANAGER_REQUEST
	804	AL_APPLICATION_REQUEST
110[22]	**State transition Stop = SUSPENDED_STOPPED_STOP** Requested by AL or SP System Manager (the one that can request this state change is only the one that has put the application in SUSPENDED state).	
	Associated Status Codes	
	000	OK
	001	TIMEOUT
	002	WRONG_STATE
	004	SOFTWARE_ERROR
	802	SP_SYSTEM_MANAGER_REQUEST
	803	AL_SYSTEM_MANAGER_REQUEST
111[22]	**State transition Stop = DISABLED_STOPPED_STOP** Requested by CUSS Application Manager or SP System Manager.	
	Associated Status Codes	
	000	OK
	001	TIMEOUT
	002	WRONG_STATE
	004	SOFTWARE_ERROR
	801	CUSS_MANAGER_REQUEST
	802	SP_SYSTEM_MANAGER_REQUEST
112[22]	**State transition Resume = SUSPENDED_AVAILABLE** Requested by SP or AL System Manager.	
	Associated Status Codes	
	000	OK
	001	TIMEOUT
	002	WRONG_STATE
	004	SOFTWARE_ERROR
	802	SP_SYSTEM_MANAGER_REQUEST
	803	AL_SYSTEM_MANAGER_REQUEST

Event Codes		
Event Code	**Description**	
113[22]	**State transition Suspend = AVAILABLE_SUSPENDED** Requested by SP or AL System Manager.	
	Associated Status Codes	
	000	OK
	001	TIMEOUT
	002	WRONG_STATE
	004	SOFTWARE_ERROR
	802	SP_SYSTEM_MANAGER_REQUEST
	803	AL_SYSTEM_MANAGER_REQUEST
114	**State transition Restart = INITIALIZE_STOPPED_RESTART** Requested by CUSS Application Manager or SP System Manager or AL application.	
	Associated Status Codes	
	801	CUSS_MANAGER_REQUEST
	802	SP_SYSTEM_MANAGER_REQUEST
115	**State transition Restart = AVAILABLE_STOPPED_RESTART** Requested by SP System Manager or CUSS Application Manager or AL application.	
	Associated Status Codes	
	801	CUSS_MANAGER_REQUEST
	802	SP_SYSTEM_MANAGER_REQUEST
	804	AL_APPLICATION_REQUEST
116	**State transition Restart = ACTIVE_STOPPED_RESTART** Requested by CUSS Application Manager or SP System Manager or AL application.	
	Associated Status Codes	
	801	CUSS_MANAGER_REQUEST
	802	SP_SYSTEM_MANAGER_REQUEST
	804	AL_APPLICATION_REQUEST
118	**State transition Restart = SUSPENDED_STOPPED_RESTART** Requested by CUSS Application Manager or SP System Manager or AL application.	
	Associated Status Codes	
	801	CUSS_MANAGER_REQUEST
	802	SP_SYSTEM_MANAGER_REQUEST
	804	AL_APPLICATION_REQUEST

[22] 8XX will be sent to the affected application as unsolicited events, and all other status codes will be used in the returned event of the directive issued by the requester (SM).

Event Codes		
Event Code	**Description**	
119[22, 23]	**State transition Load = STOPPED_INITIALIZE** Requested by CUSS Application Manager or System Manager.	
	Associated Status Codes	
	000	OK
	001	TIMEOUT
	002	WRONG_STATE
	003	CANCELLED
	004	SOFTWARE_ERROR
	303	CRITICAL_SOFTWARE_ERROR
	305	NOT_RESPONDING
	801	CUSS_MANAGER_REQUEST
	802	SP_SYSTEM_MANAGER_REQUEST
	803	AL_SYSTEM_MANAGER_REQUEST
120[22, 23]	**State transition Load = DISABLED_INITIALIZE** Requested by CUSS Application Manager or SP System Manager after human intervention occurs	
	Associated Status Codes	
	000	OK
	001	TIMEOUT
	002	WRONG_STATE
	003	CANCELLED
	004	SOFTWARE_ERROR
	303	CRITICAL_SOFTWARE_ERROR
	305	NOT_RESPONDING
	801	CUSS_MANAGER_REQUEST
	802	SP_SYSTEM_MANAGER_REQUEST
121	**State transition Restart = UNAVAILABLE_STOPPED_RESTART** Requested by CUSS Application Manager or SP System Manager or AL application.	
	Associated Status Codes	
	801	CUSS_MANAGER_REQUEST
	802	SP_SYSTEM_MANAGER_REQUEST
	804	AL_APPLICATION_REQUEST
122	**State transition Disable = UNAVAILABLE_DISABLED** Requested by CUSS Application Manager.	
	Associated Status Codes	
	303	CRITICAL_SOFTWARE_ERROR
	305	NOT_RESPONDING
	306	THRESHOLD_ERROR
	801	CUSS_MANAGER_REQUEST

[23] This is an internal platform event; the application will not yet be loaded to receive CAM events.

Event Codes		
Event Code	**Description**	
123[22]	**State transition Suspend = UNAVAILABLE_SUSPENDED** Requested by SP or AL System Manager.	
	Associated Status Codes	
	000	OK
	001	TIMEOUT
	002	WRONG_STATE
	004	SOFTWARE_ERROR
	802	SP_SYSTEM_MANAGER_REQUEST
	803	AL_SYSTEM_MANAGER_REQUEST
127[22]	**State transition Resume = SUSPENDED_UNAVAILABLE** Requested by AL application or SP System Manager.	
	Associated Status Codes	
	000	OK
	001	TIMEOUT
	002	WRONG_STATE
	004	SOFTWARE_ERROR
	802	SP_SYSTEM_MANAGER_REQUEST
	803	AL_SYSTEM_MANAGER_REQUEST
128[22]	**State transition Stop = UNAVAILABLE_STOPPED_STOP** Requested by AL application or SP/AL System Manager.	
	Associated Status Codes	
	000	OK
	001	TIMEOUT
	002	WRONG_STATE
	004	SOFTWARE_ERROR
	802	SP_SYSTEM_MANAGER_REQUEST
	803	AL_SYSTEM_MANAGER_REQUEST
	804	AL_APPLICATION_REQUEST
129	**State transition Check = INITIALIZE_UNAVAILABLE** Requested by AL application.	
	Associated Status Codes	
	804	AL_APPLICATION_REQUEST
130	**State transition Check = AVAILABLE_UNAVAILABLE** Requested by AL application.	
	Associated Status Codes	
	804	AL_APPLICATION_REQUEST
132	**State transition Wait = ACTIVE_ACTIVE** Requested by AL Application.	
	Associated Status Codes	

Event Codes		
Event Code	**Description**	
	804	AL_APPLICATION_REQUEST
133	**State transition Wait = ACTIVE_UNAVAILABLE** Requested by AL Application.	
	Associated Status Codes	
	804	AL_APPLICATION_REQUEST
Application States		
202	**State: UNAVAILABLE** No state transition. Application is in UNAVAILABLE state.	
	Associated status code	
	000	OK
	001	TIMEOUT
	002	WRONG_STATE
	004	SOFTWARE_ERROR
204	**State: STOPPED** No state transition. Application is in STOPPED state	
	Associated status code	
	000	OK
	001	TIMEOUT
	002	WRONG_STATE
	004	SOFTWARE_ERROR
206	**State: DISABLED** No state transition. Application is in DISABLED state.	
	Associated status code	
	000	OK
	001	TIMEOUT
	002	WRONG_STATE
	004	SOFTWARE_ERROR
	303	CRITICAL_SOFTWARE_ERROR
	305	NOT_RESPONDING
	306	THRESHOLD_ERROR
	310	KILL_TIMEOUT
207	**State: INITIALIZE** No state transition. Application is in INITIALIZE state.	
	Associated status code	
	000	OK
	001	TIMEOUT
	002	WRONG_STATE
	004	SOFTWARE_ERROR
208	**State: AVAILABLE** No state transition. Application is in AVAILABLE state.	

Event Codes		
Event Code	**Description**	
	Associated status code	
	000	OK
	001	TIMEOUT
	002	WRONG_STATE
	004	SOFTWARE_ERROR
209	**State: ACTIVE** No state transition. Application is in ACTIVE state.	
	Associated status code	
	000	OK
	001	TIMEOUT
	002	WRONG_STATE
	004	SOFTWARE_ERROR
	309	SESSION_TIMEOUT

3.8 Media Device Behavior and Event Sequence

This section is based on CUSS 1.0 Addendum A.1.36.

The virtual component model for CUSS devices makes it more difficult to understand how “real” device behaviour maps to the events and codes listed above. To assist in this understanding, this section explains in detail the behaviour of MediaInput devices on a CUSS kiosk (such as card and passport readers.)

Please review Chapter 7 for exhaustive details on how to find and use Media devices and other device types. This section provides an explanation of how a CUSS application should manage events from Media devices that the application has detected and acquired.

Even though information below may refer specifically to cards, it applies to any virtual device linking that corresponds to a real media device (ATB2 coupon insertion, dip or swipe passport readers, etc.)

Some behaviour is dependant on specific hardware capabilities. For example, the MEDIA_INCOMPLETELY_INSERTED event can only be published when the hardware can actually detect this condition.

3.8.1 Dip Media Reader

- User inserts on front sensor only, and then removes it within a platform/device timeout, corresponds to MEDIA_INCOMPLETELY_INSERTED.
- User inserts on front sensor, then waits (platform/device timeout) without fully inserting or removing, corresponds to MEDIA_MISPLACED.
- User fully inserts the media and the device/platform can detect that it is incorrectly placed (such as incorrect flux location for a card reader), also corresponds to MEDIA_MISPLACED.
- User inserts on front sensor and then fully inserts (within platform/device timeout) corresponds to MEDIA_PRESENT.
- If MEDIA_PRESENT is given and device/platform detects that media is faulty or damaged, issue MEDIA_DAMAGED.
- If MEDIA_PRESENT or MEDIA_MISPLACED is given and a jam is detected (if possible) issue MEDIA_JAMMED.
- If MEDIA_PRESENT is given and data is unavailable, issue DATA_MISSING, FORMAT_ERROR or LENGTH_ERROR as needed.
- If MEDIA_PRESENT if given but removed before data can be read, issue DATA_MISSING.
- Otherwise, if MEDIA_PRESENT is given and data is read, issue DATA_PRESENT
- If MEDIA_PRESENT is given and then the media is fully removed, including the front sensor, issue MEDIA_ABSENT after the data event.
- If MEDIA_MISPLACED is given and then the media is fully removed, issue MEDIA_ABSENT.
- If MEDIA_JAMMED is given and the jam is cleared, return MEDIA_ABSENT.

3.8.2 Motorized Media Reader

- The same meaning applies as for dip readers, where the “front sensor” implies the throat/gate/flux sensor that is usually in place on a motorized reader.
- Fully inserted implies that the media is ingested into the device.
- The application should handle MEDIA_MISPLACED event as it would handle MEDIA_ABSENT (ie, offer() the card back to the user for further processing.)
- Issue MEDIA_ABSENT when media is fully clear of the device (front or capture.)

3.8.3 Swipe Media Reader

- Swipe readers are quite simple devices, so will typically generated the events MEDIA_PRESENT, then DATA_PRESENT (or DATA_MISSING), then MEDIA_ABSENT, in sequence with little or no delay.

- If the hardware supports additional status and detection capabilities, it should make use of those sensors to generate events consistent with the dip or motorized reader behaviours above.

Error! Not a valid link.

Ch 4: Real Component Characteristics

This section lists all physical components (Mandatory, Recommended, and Optional) in a CUSS platform, including their hardware and software characteristics. For characteristics of the corresponding virtual components, refer to Section Ch 5: (Virtual Component Characteristics). This section includes the following subsections:

Mandatory Components

Recommended Components

Optional Components

Application developers should also review Chapter 7, which provides much of the same information as here, but organized more practically to assist in writing code to find and use CUSS devices in a kiosk application.

4.1 Mandatory Components – All Kiosks

All CUSS kiosks must include common virtual components representing the following devices:

Clock

CUSS

Enclosure

Display

Hard Disk

Network

System

Touch Screen Overlay

Mandatory Devices for Check-in Kiosks:

CUSS kiosks classified as check-in kiosks must include the following physical devices and associated CUSS virtual component devices. Unless otherwise specified, platform and application providers shall understand that a CUSS kiosk is assumed to be a Check-in Kiosk:

Boarding Pass Printer (AEA)

Magnetic Card Reader

Barcode Scanner, 2D-capable

Mandatory Devices for Self Bag Drop Kiosks

CUSS kiosks classified as Self Bag Drop (SBD) kiosks must include the following physical devices and associated CUSS virtual component devices:

Self Bag Drop Coneyor with Scale and LPN-capable Barcode Scanner

Barcode Scanner, 2D-capable

Mandatory Devices for Other Kiosks

Other kiosks may be deployed that comply with the CUSS standard that are not Check-in Kiosks or Self Bag Drop kiosks. These are understood to be specialized self-service devices that are not intended as general passenger check-in machines.

Platform providers that choose to deploy CUSS kiosks that are not for Check-in or Self Bag Drop should clearly indicate the intended purposes of these kiosks and there limitations.

Further, platform providers who choose to deploy non-check-in kiosks should have no expectation that generally-available CUSS check in applictions from airline providers will operate or be modified to operation on non-check-in CUSS kiosks.

Examples of these other types of kiosks could be:

- Document Scanning device
- Dedicated printing station
- Flight rebooking or baggage recovery kiosks

4.1.1 Boarding Pass Printer (AEA)

One printer capable of print boarding passes using the AEA standard is required. This can be a GPP or an ATB. For additional information and guidelines on using AEA and stock characteristics, please also review Appendices G and H.

Hardware Characteristics	
Technology	Any with good printing quality, to be defined in the SLA between platform provider and service provider or application provider. AEA requires 200dpi or greater for PCX logo printing.
Speed	As defined in the SLA between platform provider and service provider or application provider.
Bin	At least one

Hardware Characteristics	
Printing capability	Landscape/portrait Bar Code bar Code39, 128 and 2-of-5 and all its subsets in addition to IATA RP 1720a Attachment F Single color Dot addressable graphics 200 dpi or more (to comply with IATA Resolution 792 for BCBP) PECTAB support; Multiple PECTAB context[29]; if the printer is not able to support PECTAB then the CUSS interface must emulate this functionality **For GPP only** Support ANSI Latin 1 character set Unicode Character set Support at least the following 6 fonts: 10 cpi, 10 cpi large, 17 cpi condensed, 17 cpi condensed and bold, OCR_B, 5 cpi **For ATB only** Support of ASCII character set Fonts: 10 cpi, 10 cpi large, 17 cpi condensed, 17 cpi condensed and bold, OCR_B, 5 cpi, as defined by IATA 722c and 722d
Cutting/Bursting	Must be done prior to user accessibility to printed document
Implementation	1 **MediaOutput** virtual component per Feeder component 1 **Feeder** virtual component per stock type if more than one stock type is installed in the printer bins 1 **Dispenser** virtual component

Stock Characteristics	
Media type	As per IATA RP 1706e, IATA Resolutions 722c and 722e Blank paper Direct thermal (for GPP only) Form Fanfold Roll Detached form
Dimension	As per IATA RP 1706d Length: 203.20 mm (8 inches) Width: 82.55 mm (3.250 inches) Perforation: (from right edge) **ATB**: 50.8 mm (2 inches) **GPP** [30]: 88.9 mm (3.5 inches)

4.1.2 Clock

Software Characteristics	
Description	Software that represents the hardware clock set at local time
Implementation	**DataInput** virtual component
Data Definition	CLOCK data type as defined in Section 3.1.9 Data:

[29] At Airline application launch, CUSS platform must, if supported by the printer, load all required PECTAB for this application in parallel to the application caching. CUSS platform does the PECTAB context switching when the application becomes ACTIVE. This allows getting airline PECTAB without waiting time impact. PECTAB must conform to AEA standard

[30] The platform will not modify data streams to accommodate the perforation. (e.g. AEA data expecting a 2 inches perforation printed on GPP with 3.5 inches perforation).

Software Characteristics	
	yyymmddhhmmss local time

Note: The use of the hardware clock is reserved for CUSS component interface and/or CUSS application manager. Any application trying to set the hardware clock will be flagged as having a bad behavior.

4.1.3 CUSS

Software Characteristics	
CUSS Application Manager	Software that manages the CUSS applications and environment on the kiosk
Device Components	Software that interfaces with peripherals
Logging Services[31]	Software that handles the log (system, functional and business)
System Manager Interface	Software that implements the interface between SP/AL System Manager and CUSS platform
Common Launch Application	Application that will be used in idle state of the kiosk to presents attract loop and self-service application selection

4.1.4 Enclosure

Hardware Characteristics	
Power supply	Inside the casing
Casing	As per IATA RP 1706c (section 11.3)
Implementation	No associated virtual component

[31] CUSS Logging Services is not specified in CUSS 1.0. This is left up to the platform provider.

4.1.5 Display

Hardware Characteristics	
Technology	Display of any type
Character set	ISO 8859-1 Latin 1 Unicode
Physical dimension	Diagonal 30.48 cm / 12 inches or higher
Dimension ratio height/wide	¾
Resolution	Must support at least one of the following resolutions, in pixels: 1024*768, or at least 1024 pixels wide and at least 768 pixels high 1280*1024 1600*1200
Color	16 bits or higher
Refresh rate CRT/LCD	85 or higher/60
Implementation	Direct access by the application NativeDevice class: **Display**; for event generation and query only

Note: Currently only one screen resolution should be used if the touch screen overlay device is not automatically recalibrated.

4.1.6 Hard Disk

Hardware Characteristics	
Technology	According to available technology
Space	1024MB per self-service application (mainly for configuration file that can be location dependant
Implementation	Direct access by the application NativeDevice class: **Storage**; for event generation and query only

4.1.7 Magnetic Card Reader

Hardware Characteristics	
Mechanism	Any available on the market Recommended: Dip, swipe, or motorized
Character set	ISO 8859-1 Latin 1 Unicode
Read capability	Magnetic card reader must be able to read from track 1 to 3 or any combination of these tracks. **Please review Chapter 8 for Payment Data handling requirements.**
Protection	Security gate (Recommended)
Implementation	**MediaInput** virtual component **Dispenser** virtual component (for motorized readers only)

Stock Characteristics	
Media type	Magnetic stripped card, ISO standard, low coercivity[32]
Dimension	As per ISO standard (7810, 7811, 7812)

[32] Please read Section 1.7 for guidelines about properly handling card data

4.1.8 Network

Hardware Characteristics	
Technology	According to available technology
Protocol	TCP/IP
Connectivity	Platform Provider discretion
Implementation	Direct access by the application NativeDevice class: **Network**; for event generation and query only

4.1.9 System

This includes a computer and its associated system software. The requirements are based on the minimum specifications required to support the technologies listed in Appendix D.

Hardware Characteristics	
Speed	Pentium III 1Ghz or better (or equivalent)
Memory	512MB or higher including 128 MB per CUSS application
Cache	256 KB or higher Recommended: 512 KB or higher
Removable media	USB or CD/DVD-ROM recommended
Keyboard and Mouse	Recommended: 1 adapter for each (inside the casing)
Implementation	No associated virtual component

Software Characteristics	
Operating System	Any one that supports the minimum and recommended components
Browser	A standard, commercially available browser that supports the current standard as officially released by W3C. CUSS 1.3 requires that this browser be the Standard CUSS Browser.
CORBA environment	Any ORB that implements CORBA 2.3 or higher with corbaloc support
Java environment and other technologies	Please see Appendix D for the list of Java and other software technologies required on a CUSS kiosk. CUSS 1.3 requires that this browser by the Standard CUSS Java.
Character set	ISO 8859-1 Latin 1 Unicode character set
Implementation	No associated virtual component

4.1.10 Touch Screen Overlay

Hardware Characteristics	
Technology	Touch screen overlay of any type
Implementation	Direct access by the application NativeDevice class: **UserInput**; for event generation and query only

4.1.11 Barcode Scanner with 2D support

Hardware Characteristics	
Technology	Any barcode scanner capable of reading 1D linear and 2D barcodes listed in IATA Resolution 792.
Speed	As defined in the SLA between platform provider and service provider or application provider.
Scanning capability	All barcode types listed in IATA Resolution 792. 2D symbologies:, • PDF417 • QRCode • Aztec • Datamatrix Common 1D symbologies (Code39, Barcode 128, Interleaved 2of5, UPC)
Implementation	1 **MediaInput** virtual component with the following characteristics in order to differentiate it from a passport reader: ComponentFonts.BarcodeStandard.Code39 ComponentFonts.BarcodeStandard.Code128 ComponentFonts.BarcodeStandard.Code2of5

4.1.12 Self Bag Drop device

Hardware Characteristics	
Technology	Any integrated conveyor system that integrates bag acceptance, weighing, and scanning technologies.
Speed	As defined in the SLA between platform provider and service provider or application provider.
Scanning capability	Interleaved 2of5 IATA License Plate Number barcodes:
Implementation	Virtual component implementation of the AEA-SBD interface. Virtual component implement of the CUSS-SBD interface. See Chapter 7 for details.

4.2 Recommended Components

This section gives the list of all recommended components for a CUSS platform. None of them are required but a typical CUSS platform could have some of them. Recommended components are:

Bag Tag Printer

Door Sensor

Hardware Watch Dog

Passport Reader

Receipt Printer

UPS

ATB2 Device

4.2.1 ATB2 Device

In replacement of Boarding Pass Printer in Section 0.

Hardware Characteristics	
Speed	As defined in the SLA between platform provider and service provider or application provider.
Printing capability	Magnetic encoding track 1 to 4 Landscape/portrait Bar Code bar Code39, 128 and 2-of-5 and all its subsets in addition to IATA RP 1720a Attachment F, Single color Dot addressable graphics 200 dpi or more (to comply with IATA Resolution 792 for BCBP) PECTAB support; Multiple PECTAB context[33]; if the printer is not able to support PECTAB then the CUSS interface must emulate this functionality Support of ASCII character set Fonts 10 cpi, 10 cpi large, 17 cpi condensed, 17 cpi condensed and bold, OCR_B, 5 cpi, as defined by IATA 722c and 722d
Implementation	1 **MediaInput** virtual component for the coupon reading 1 **MediaOutput** virtual component for writing on the inserted coupon 1 **MediaOutput** virtual component per Feeder component 1 **Feeder** virtual component per stock type if more than one stock type is installed in the printer bins 1 **Dispenser** virtual component (even if an escrow is not installed) 1 **Dispenser** virtual component for escrow (if installed) 1 **Capture** virtual component for the capture bin (if installed) 1 **Capture** virtual component for the void bin (if installed)

[33] At Airline application launch, CUSS platform must, if supported by the printer, load all required PECTAB for this application in parallel to the application caching. CUSS platform does the PECTAB context switching when the application becomes ACTIVE. This allows getting airline PECTAB without waiting time impact. PECTAB must conform to AEA standard.

Stock Characteristics	
Stock type	Blank ATB Card Type 4 stock with a magnetic stripe and without a binding stub, as per IATA Resolution 722e.
Dimension	As per IATA Resolution 722c.

4.2.2 Bag Tag Printer

Hardware Characteristics	
Technology	Any with good printing quality, to be defined in the SLA between platform provider and service provider or application provider.
Speed	As defined in the SLA between platform provider and service provider or application provider.
Printing capability	Landscape, portrait Bar code as per IATA Resolution 740 Attachment B Single or multiple color Dot addressable graphics (for logo) PECTAB support, multiple PECTAB contexts[34]. If the printer does not support PECTAB, this functionality must be emulated by the CUSS interface. Support of ASCII character set Fonts as defined by AEA and IATA
Cutting/Bursting	Optionally done prior to user accessibility to printed document
Implementation	1 **MediaOutput** virtual component per Feeder component 1 **Feeder** virtual component per stock type 1 **Dispenser** virtual component (even if an escrow is not installed) 1 **Dispenser** virtual component for escrow (if installed) 1 **Capture** virtual component for the capture bin (if installed) 1 **Capture** virtual component for the void bin (if installed)

Stock Characteristics	
Stock type	As per IATA Resolution 740 Blank paper Form Roll Fanfold Detached form
Dimension	As per IATA Resolution 740 Section 8, which contains CUSS-specific bagTag information specification

[34] At Airline application launch, CUSS platform must, if supported by the printer, load all required PECTAB for this application in parallel to the application caching. CUSS platform does the context switching when the application becomes ACTIVE. This allows getting airline PECTAB without waiting time impact. PECTAB must conform to AEA standard.

4.2.3 Door Sensor

Hardware Characteristics	
Technology	According to available technology
Location	Inside the casing
Implementation	**DataInput** virtual component

Software Characteristics	
Data Definition	CLOCK data type as defined in Section 3.1.9 Data: OPEN for Door open CLOSED for Door closed UNKNOWN for Sensor failure (unknown status)

4.2.4 Hardware Watch Dog[35]

Hardware Characteristics	
Technology	According to available technology (PC controlled and hardware implementation required
Implementation	NativeDevice class: **DataInput**; for event generation and query only; also used for logging usage (the system must log each time it goes down properly and each time it goes up; this is true as well for the CUSS Application Manager itself)

4.2.5 Passport Reader

Hardware Characteristics	
Mechanism	Swipe or dip
Character set	ISO 8859-1 Latin 1 Unicode
Read capability	OCR-A PECTAB support (optional), if the application uses a PECTAB and the reader does not supports PECTAB, the CUSS interface must emulate this functionality
Implementation	**MediaInput** virtual component

Stock Characteristics	
Passport type	All OCR encoded passport

[35] The software watch dog component (virtual watch dog component) was rejected by vote.

4.2.6 Receipt Printer[36]

Hardware Characteristics	
Technology	Any with good printing quality, to be defined in the SLA between platform provider and service provider or application provider.
Speed	As defined in the SLA between platform provider and service provider or application provider.
Printing capability	Landscape, portrait Bar Code bar Code39, 128 and 2-of-5 and all its subsets in addition to IATA RP 1720a Attachment F Single color Dot addressable graphics (for logo) 200 dpi or more (to comply with IATA Resolution 792 for BCBP) PECTAB support (optional); if the application uses a PECTAB and the printer does not support PECTAB, this functionality must be emulated by the CUSS interface **For GPP only** Support ANSI Latin 1 character set Unicode Character set Support at least the following 6 fonts: 10 cpi, 10 cpi large, 17 cpi condensed, 17 cpi condensed and bold, OCR_B, 5 cpi **For ATB only** Support of ASCII character set Fonts: 10 cpi, 10 cpi large, 17 cpi condensed, 17 cpi condensed and bold, OCR_B, 5 cpi, as defined by IATA 722c and 722d
Cutting/Bursting	Must be done prior to user accessibility to printed document
Implementation	1 **MediaOutput** virtual component per Feeder component 1 **Feeder** virtual component per stock type if more than one stock type are installed in the printer bins 1 **Dispenser** virtual component (even if an escrow is not installed) 1 **Dispenser** virtual component for escrow (if installed) 1 **Capture** virtual component for the capture bin (if installed) 1 **Capture** virtual component for the void bin (if installed)

Stock Characteristics	
Stock type	As per IATA Resolution 1706d Blank paper Form Roll FanFold Detached form
Dimension	Height: 80 mm minimum Length: variable

[36] A receipt can be printed on a blank boarding pass.

4.2.7 UPS

Hardware Characteristics	
Technology	According to available technology (PC controlled required)
Duration	The required time to do an orderly shutdown of the kiosk when power supply fails (5 minutes minimum).
Implementation	**DataInput** virtual component
Data Definition	
Battery	Power supply down Battery low
Normal	Battery charged Power supply up
Power	Power supply down Battery charged
Empty	Battery empty Power supply up

4.3 Optional Components

This section gives a list of optional components for a CUSS platform that are used by a restricted number of airlines. This is not an exhaustive list of all possible components on a CUSS platform, and the list is merely provided as an example.

None of the components on this listed are required but these components can be installed as based for a specific airport implementation as well as on specific airline request.

Note that some devices are secured (e.g. by key lock or PC controlled access). This feature will not be shown in the table below but rather in the component characteristics (Refer to Section Ch 5:: Virtual Component Characteristics) of the applicable devices.

Optional Components		
Real Component	**Description**	**Media**
Audio	Sound card and speakers with adjustable volume	Nil
Bills capture bin	Box to capture bills	Bills country dependant
Bills dispenser	Device that give bills to the customer	Bills, country dependant
Bills reader	Device that read the bill value	Bills , country dependant
Boarding pass dispenser bin	Device that supply boarding passes	Boarding passes as per IATA standard
Boarding pass capture bin	Box to capture Boarding passes printed or read	Boarding passes as per IATA standard
Chip card capture bin	Box to capture chip card	Chip card as per ISO 7816 & EMV 1.3.2 standard
Chip Card Device	Contact	Chip Card as per ISO 7816 & EMV 1.3.2 standard

Optional Components		
Real Component	**Description**	**Media**
Chip card dispenser bin	Box to supply chip card	Chip card as per ISO 7816 & EMV 1.3.2 standard
Chip Card Reader	Contact	Chip Card as per ISO 7816 & EMV 1.3.2 standard
Chip Card Writer	Contact	Chip Card as per ISO 7816 & EMV 1.3.2 standard
Coins capture bin	Box to capture coins	Coins, country dependant
Coins dispenser	Device that give coins to the customer	Coins, country dependant
Coins reader	Device that read the coins value	Coins, country dependant
Device Sentry	A device that controls power supply of other component	Nil
Digital I/O	Digital/Analog conversion card with device attached to it or Digital card with digital device attached	Nil
Escrow	A PC controlled escrow	Boarding pass as IATA standard
Fingerprint reader	Device to read human fingerprints	Nil
GPP Printer	General Purpose Printer	Paper as per current standard A4 or 8.5*11, and/or as needed for specialty documents.
Hand reader	Device to read the hand shape	Nil
Iris Scanner	Biometrics device to scan the eye iris	Nil
Keypad		Nil
LED Indicator		Nil
Magnetic card capture bin	Box to capture magnetic card	Magnetic card as per ISO standard
Magnetic Card Device	Reader and encoder	Magnetic Card as per ISO standard
Magnetic card dispenser bin	Box to supply magnetic card	Magnetic card as per ISO standard
Magnetic Card encoder		Magnetic Card as per ISO standard
Media sensor	Device to detect the presence/absence or the level of media	Any kind of media handled by the associate device
Motion detector		Nil
OCR reader		Paper with text
PIN Pad	Pin Pad block	Nil
Proximity Sensor	Change state when leaving	Nil
RF ID	Radio Frequency reader (e.g. RF contactless smartcard)	RF card or other RF media
Retina or other biometric reader		Nil
Secure Enclosure	PC controlled enclosure lock	Nil
Speaker		Nil
Ticket printer		TAT as per IATA standard
Video Camera	A PC controlled video camera	Nil
Visual Customer Assistance Light	Bigger than LED indicator	Nil
Weight Scale		Nil

Any printer device in the kiosk must meet the IATA Resolution 792 for Bar-Coded Boarding Pass requirements by offering 200dpi or better print resolution.

Ch 5: Virtual Component Characteristics

This section briefly describes the characteristics of the various virtual components that make-up the CUSS Platform. Some of these characteristics may or may not be applicable depending on the physical device that is being represented by the corresponding virtual component(s). In addition, most of these component characteristics are read-only. For those characteristics that could be set by a CUSS application, a corresponding "set" directive has been provided.
For more detailed information, please refer to the "characteristics.idl" (Appendix C) file provided as part of the CUSS Technical Specifications.
This section includes the following subsections:

Common Characteristics

Application Characteristics

Capture Characteristics

DataInput Characteristics

DataOutput Characteristics

Dispenser Characteristics

Note 1 **(CUSS 1.0 Addendum A.1.44):**

Certain types of devices can include a Dispenser component that does not actual present its media to the user. These devices are internal "feeders" but incorrectly qualified as Dispensers within this CUSS standard. An example of such a device is the Dispenser associated with the insertion slot of an ATB2 reader/printer, which ejects the inserted coupon into an escrow device.

The platform must identify any of these "userless" Dispensers so that the application can interact with them correctly (for example, the application does not need to call the offer() directive for a userless component.)

Any virtual component in a CUSS device linking that is classified as a Dispenser but does not interact with the user (ie, dispenses/feeds to another part of the device) must include in its firmwareVersion characteristics string the following string:

FEEDER_USERLESS

Display Characteristics

Feeder Characteristics

MediaInput Characteristics

MediaOutput Characteristics

Network Characteristics

Storage Characteristics

UserInput Characteristics

UserOutput Characteristics

Application developers should also review Chapter 7, which provides much of the same information as here, but organized more practically to assist in writing code to find and use CUSS devices in a kiosk application.

Note: Some attributes may not be applicable. This depends on the corresponding real component (e.g. *IOMode* apply doesn't apply for GPP MediaOutput and *timeZone* doesn't apply to door sensor data input) and whether or not the corresponding physical attribute is actually supported by the real component (e.g. if a stock sensor or a document counter is present or not).

Note 1:
After the original CUSS 1.0 specification was released, it became apparent that the Interface Characteristics fields were not enough to convey all types of information needed by CUSS applications. For this reason, some Characteristics fields are being reused and "overloaded" to contain additional details. Please review Section 5.15 for a list of cases where this approach has been defined.

Note (CUSS 1.0 Addendum A.1.44):
Certain types of devices can include a Dispenser component that does not actual present its media to the user. These devices are internal "feeders" but incorrectly qualified as Dispensers within this CUSS standard. An example of such a device is the Dispenser associated with the insertion slot of an ATB2 reader/printer, which ejects the inserted coupon into an escrow device.

The platform must identify any of these "userless" Dispensers so that the application can interact with them correctly (for example, the application does not need to call the offer() directive for a userless component.)

Any virtual component in a CUSS device linking that is classified as a Dispenser but does not interact with the user (ie, dispenses/feeds to another part of the device) must include in its firmwareVersion characteristics string the following string:

FEEDER_USERLESS

Non-applicable values are either represented as:
–1 (for attributes of type integer or string) or
nonApplicableValue (for attributes of enumerated types).

5.1 Common Characteristics

The following Characteristics are shared by some characteristics outlined in Sections 5.2 through 5.14.

5.1.1 Bin Settings

Attribute	Description
`BinSize`	Describes the maximum number of documents a bin can hold. This corresponds to the MEDIA_FULL status.
`AlmostFullLevel`	Shows high threshold of the bin: the number of documents correspondent to MEDIA_HIGH status (e.g. for a Capture component), if corresponding sensor is installed.
`AlmostEmptyLevel`	Shows low threshold of the bin: the number of documents correspondent to MEDIA_LOW status (e.g. for a Feeder component), if corresponding sensor is installed
`currentNoOfDocuments`	Shows the current number of documents in the bin, if document counter is present. This number is not always guaranteed to be accurate (e.g. AlmostEmptylevel has not been reached for a feeder component). On the other hand, when AlmostEmptyLevel is reached, the counter should reflect the best count possible (This depends on tolerance of stock thickness and assumes no human manipulation of document counter)

Note 1 (CUSS 1.0 Addendum A.1.38):
The platform must accurately set these characteristics for any Dispenser component where real capacity limits exist. This allows a CUSS application to accurately manage multiple-document jobs.

5.1.2 ComponentFonts

Attribute	Description
`usedStandard`	Specifies which barcode standards are used (code39 or code128 or code2of 5) if applicable
`Fonts`	List of all Fonts (and their attributes) available from this component. Font attributes are: fontName, fontSizes, vectorFont, bold, italic, underlined, strikethrough, reverse, superscript, subscript, colorDepth, spacing, and CharacterLength.

5.1.3 IOMode

Attribute	Description
`mode`	The currently used mode for reading/writing (Check-in or Revalidation for ATB) if applicable

5.1.3.1 setIOMode

Description	Set the input/output mode for ATB printers.
Apply to	All acquired virtual components of class "MediaInput" or "MediaOutput"
Available	AL application (in INITIALIZE or ACTIVE state)
Access	Exclusive, local/remote, synchronous
Structure sent	Application Token: A valid active application reference, defined in Section 3.1.4 InputOutputMode: The input/output mode to be used (check-in or Revalidation)
Structure returned	ReturnCode, defined in Appendix A

Note: `setIOMode` directive is context-specific; when the application becomes ACTIVE, the platform ensure to select the proper context of that application.

5.1.4 Location

Attribute	Description
`Map`	URL to location image file of a kiosk component
`mapType`	The type of the image (BMP, GIF, JPEG, PNG, Flash, etc) if applicable
`howTo`	URL to usage image/animation file of a kiosk component
howToType	The type of the image/animation (GIF, JPEG, Flash etc) if applicable
`componentLocation`	Where to find the component (Inside or outside the kiosk)

5.1.5 Manufacturer

Attribute	Description
`realComponentIdentification`	Component identification for use by system manager. (e.g. ATB-Printer-BIN1)
`downloadableFirmware`	Describes whether the firmware can be updated or not.
`firmwareVersion`	Version of firmware/software.
`manufacturerName`	Name of manufacturer.
`modelNumber`	Model number of hardware component.
`serialNumber`	Serial number of hardware component.

5.1.6 MediaType

Attribute	Description
`type`	Attribute containing one media type of the following: MagneticStripe, Chip, Printed (OCR/Barcode/Plain paper), JIS

5.1.7 MediaTypeList

Attribute	Description
mtList	List of media types. (Refer to Section 5.1.6: MediaType)

5.2 Application Characteristics

Please refer to the Section 5.1.5: Manufacturer attributes as well as the attributes listed below.

Attribute	Description
identification	Kiosk Application identification.
allContacts	The list of all available company contacts each including: company name, person name, address (postal, phone, fax, pager, e-mail, or remote support application address, if applicable) and an unspecified note on person or company.
frstIPPort	Specifies the first of IP-Port range that can be used by this application.
lastIPPort	Specifies the last of the IP-Port range that can be used by this application.

5.3 Capture Characteristics

Please refer to the Section 5.1.1: Bin and the Section 5.1.5: Manufacturer attributes.

5.4 DataInput Characteristics

Please refer to the 5.1.5: Manufacturer attributes as well as the attributes listed below.

Attribute	Description
timeZone	The time difference in hours relative to GMT. (This is applicable to Clock component)
supportedDataTypes[1]	The list of data types supported by this component (CLOCK, MSG, NIL, SWITCH)

5.5 DataOutput Characteristics

Please refer to the Section 5.1.5: Manufacturer attributes as well as the attribute listed below.

Attribute	Description
supportedDataTypes[2]	The list of data types supported by this component (MSG, NIL, SWITCH)

[1] DataInput::supportedDataTypes is not included in CUSS 1.0 IDL.
[2] DataOutput::supportedDataTypes is not included in CUSS 1.0 IDL.

5.6 Dispenser Characteristics

Please refer to the Section 5.1.1: Bin, the Section 5.1.4: Location, and the Section 5.1.5: Manufacturer attributes as well as the attribute listed below.

Attribute	Description
`kind`[3]	Specifies whether it is a real dispenser or virtual one. A dispenser component is **real** if an offer() directive is mandatory to make the document(s) available to the user (such as ejecting a card from a motorized reader, or opening an escrow door). A dispenser component is **virtual** if documents are available to the user immediately and without a call to offer() such as with a paper output chute. Both types of dispenser may or may not have sensors to detect if a document is present in the dispenser. Please review Section 3.6.9.1 offer() for more information on how to use this Characteristic and the Dispenser component status indicators to properly present documents to the user and to monitor (where possible) if the documents have been taken.

Note 1 (CUSS 1.0 Addendum A.1.44):

Certain types of devices can include a Dispenser component that does not actual present its media to the user. These devices are internal “feeders” but incorrectly qualified as Dispensers within this CUSS standard. An example of such a device is the Dispenser associated with the insertion slot of an ATB2 reader/printer, which ejects the inserted coupon into an escrow device.

The platform must identify any of these “userless” Dispensers so that the application can interact with them correctly (for example, the application does not need to call the offer() directive for a userless component.)

Any virtual component in a CUSS device linking that is classified as a Dispenser but does not interact with the user (ie, dispenses/feeds to another part of the device) must include in its firmwareVersion characteristics string the following string:

FEEDER_USERLESS

5.7 Display Characteristics

Please refer to the Section 5.1.4: Location and the Section 5.1.5: Manufacturer attributes as well as the attributes listed below.

Attribute	Description

[3] Clarification taken from Addendum A.1.11.

displayResolution	List of supported screen resolutions: 1024 indicates a resolution of 1024 by 768 1280 indicates a resolution of 1280 by 1024 1600 indicates a resolution of 1600 by 1200. Note: Currently only one screen resolution should be used if the touch screen overlay device is not automatically recalibrated. It is recommened that application suppliers use native programming methods to detect the current resolution of the Display device. It is acceptable for a CUSS platform to provide any screen resolution that is at least 1024 pixels wide and at least 768 pixels high, including portrait mode displays and displays in aspect ratios other than 4:3.
currentResolution	Currently used screen resolution.
screenDiagonal	Physical screen size measured in Millimeters.

5.7.1 setScreenResolution

Description	Sets a new resolution for the display.
Apply to	All acquired virtual components of class “Display”
Available	AL application in ACTIVE state
Access	Exclusive, local/remote, synchronous
Structure sent	Application Token: A valid active application reference, defined in Section 3.1.4 Resolution: The screen resolution to be used by the application
Structure returned	ReturnCode, defined in Appendix A

Note: **`setScreenResolution`** should return RC_NOT_SUPPORTED if not implemented by the platform.

5.8 Feeder Characteristics

Please refer to the Section 5.1.4: Location and the Section 5.1.5: Manufacturer attributes.

5.9 MediaInput Characteristics

Please refer to the Section 5.1.1: ComopnentFonts, the Section 5.1.3: IOMode, the Section 5.1.4: Location, the Section 5.1.5: Manufacturer, and the Section 5.1.7: MediaTypeList attributes as well as the attributes listed below.

Attribute	Description

typeOfReader	The kind of reader that is handled by this component (Motorized, DIP, Swipe, Contactless, FlatbedScan, PenScan).
supportedDataTypes	The list of data types supported by this component (AEA, MSG, NIL)
setupDataType[4]	Describes the type of data stream that is supported by this component.
numberOfTracks	The number of tracks that can be read by the components.

5.10 MediaOutput Characteristics

Please refer to the Section 5.1.1: ComopnentFonts, the Section 5.1.3: IOMode, the Section 5.1.4: Location, the Section 5.1.5: Manufacturer, and the Section 5.1.7: MediaTypeList attributes as well as the attributes listed below.

Attribute	Description
type	Attribute containing type of media used (Ticket, BoardingPass, GeneralPurposeDoc, BaggageTag, InsertedDoc, Card)
supportedDataTypes	The list of data types supported by this component (AEA, MSG, NIL, SVG)
bufferSize	Size of the internal data buffer.
numberOfTracks	The number of tracks that can be written by the components.
minDocumentLength	The minimum length of a document measured in Millimeters.
maxDocumentLength	The maximum length of a document measured in Millimeters.
maxPrintSizeX	The maximum printing size in X direction measured in Millimeters.
maxPrintSizeY	The maximum printing size in Y direction measured in Millimeters.
mediaTransferType[5]	Specification of printing technology used: DirectThermal or ThermalTransfer (ribbon-based) or nonApplicable (for non-printer devices; e.g. card writer)
printOrientation	The current print orientation (Portrait or Landscape)

5.10.1 setPrintOrientation

Description	Sets the printing orientation to be used by this component.
Apply to	All acquired virtual component of class “MediaOutput”
Available	AL application in ACTIVE state
Access	Exclusive, local/remote, synchronous
Structure sent	Application Token: A valid active application reference, defined in Section 3.1.14 Orientation: The printing orientation (Portrait or Landscape)
Structure returned	ReturnCode, defined in Appendix A

[4] MediaInput::setupDataType is not used in CUSS 1.0.

[5] In CUSS 1.0 IDL, MediaOutput::mediaTransferType attribute is missing. Instead, its value (DirectThermal or ThermalTransfer) should be inserted somewhere inside the string representing Manufacturuer::ModelNumber attribute.

Note: The application must check the print orientation (and set it if necessary) every time it becomes ACTIVE to guarantee the correct orientation (a previously active application might have changed the orientation).

5.11 Network Characteristics

Please refer to the Section 5.1.5: Manufacturer attributes.

5.12 Storage Characteristics

Please refer to the Section 5.1.5: Manufacturer attributes as well as the attributes listed below.

Attribute	Description
`Size`	Specifies the total size (in MB) available for an application on a disk.
`Path`	Specifies the complete path to writeable/readable location (all path specifications end with a separator, e.g. slash or backslash). For example, under Windows, the Path would be something like C:\CUSS\APPS\AC\CKC\.

In CUSS 1.2, the Path attribute will give the full path of the location that contains the application's local files. If the application has no local files, it will point to a directory created by the platform on the kiosk reserved for use by that application.

CUSS 1.2 platforms must create and provide Storage components for each application in a fashion that allows each application to have its own unique Path characteristic. The storage location can either be local to the kiosk or on a network, but the directory must be unique to the kiosk application and the specific kiosk (not shared with the same application on other kiosks.)

(For reference, this behaviour is changed from the original CUSS 1.0 Technical Specification, where the path component was shared between all applications.)

5.13 UserInput Characteristics

Please refer to the Section 5.1.4: Location and the Section 5.1.5: Manufacturer attributes.

5.14 UserOutput Characteristics

Please refer to the Section 5.1.4: Location and the Section 5.1.5: Manufacturer attributes.

5.15 Free-form Characteristics Settings

After the original CUSS 1.0 specification was released, it became apparent that the Interface Characteristics fields were not enough to convey all types of information needed by CUSS

applications and kiosks. For this reason, some Characteristics text fields are being reused and "overloaded" to contain additional details.

This section provides a summary of where such "extra settings" have been defined. For details on why each setting is used, please refer back to the original section indicated in the table. Applications that need to detect if an override is used should use case-insensitive substring matching.

Purpose	Device	Section	Characteristic	Used Strings
Find paper perforation location	ATB and GPP printers	Appendix D	Manufacturer::modelNumber (Feeder)	Perf=2 Perf=3.5
Check which version of AEA is supported by the printer	ATB and bag tag printers	Appendix D	Manufacturer::firmwareVersion (MediaOutput)	AEA2009 [for CUSS 1.3] AEA2008 [for CUSS 1.2] AEA1999 AEA2002
Identify different types or qualities of Boarding pass stock	ATB printers	Appendix G	Manufacturer::modelNumber (Feeder)	ECONOMY BUSINESS FIRST WALLET
Determine the thermal printing capabilities of an ATB2 printer	ATB printers	Appendix G	Manufacturer::modelNumber (MediaOutput)	DirectThermal ThermalTransfer
Does the AEA bagtag printer support color printing	AEA bagtag printers	Appendix D	Manufacturer::firmwareVersion (MediaOutput)	ZSOK00 ZSOK010203 [etc]
Can a barcode scanner read certain extended barcode types (2D, etc), as distinguished from another reader on the kiosk that cannot?[1]	Barcode scanners	Chapter 6 Appendix H	Manufacturer::firmwareVersion (MediaInput)	DS_TYPES_SCAN_PDF417 DS_TYPES_SCAN_* [etc] (can include multiple, see Appendix H)
Is a Dispenser component internal and userless (offer() not required)	ATB Printers with Escrow	Section 5.6	Manufacturer::firmwareVersion (Dispenser)	FEEDER_USERLESS
See if the documents being printed have a secure/control number	Printers	Appendix G	Manufacturer::serialNumber (Feeder)	CTRL:<document id> CTRL:-1

[1] CUSS 1.2 kiosks must include a barcode scanning device capable of reading PDF417 and other IATA Resolution 792 barcodes.

Purpose	Device	Section	Characteristic	Used Strings
Determine if a reader device supports any extended media type(s)	Media readers (OCR or flatbed scanners, card readers, etc.)	Chapter 6 Appendix H	Manufacturer::firmwareVersion (MediaInput)	DS_TYPES_IMAGE_IR DS_TYPES_JIS2 [etc] (can include multiple, see Appendix H)
Determine if a writer device can produce media with any extended media type(s)	Writer devices (card encoders, printers, etc.)	Chapter 6 Appendix H	Manufacturer::firmwareVersion (MediaOutput)	DS_TYPES_SAFLOK DS_TYPES_TIMELOX [etc] (can include multiple, see Appendix H)
Find exact vendor version of platform	Environment	Section 3.3.1.1	EnvironmentLevel::cussInterfaceVersionMax	<platform version string>
Determine if a kiosk is offsite (not at an airport)	Environment	Section 3.3.1.2	EnvironmentLevel::kioskLocation	OFFSITE
Does a printer support PDF printing	GPP Printers	Section 6.4.2	Manufacturer::firmwareVersion (MediaOutput)	DS_TYPES_PRINT_PDF
Does a printer support 2-sided printing	GPP Printers	Section 6.4.3	Manufacturer::firmwareVersion (MediaOutput)	TWOSIDED

Ch 6: Extended Device & Media Type Handling

The original CUSS 1.0 standard was designed to operate with the "normal" self-service devices on a kiosk, such as an ATB2 or GPP printer, or an ISO magnetic card reader. However, many new devices used in kiosks blur the line between these typical devices, and combine multiple functions, data types, and operations.

- Integrated document scanner that supports multiple formats:
  - MRZ and embedded barcode decoding
  - Visible and non-visible imaging
  - RFID reading and updating
- Card readers/writers that include non-ISO formatting
  - JIS2 or magnetic lock formats
  - Integrated chip card support
  - Optical scanning an imaging support
- Barcode scanners that support new types of data
  - 2D barcode such as DataMatrix, Aztec, etc

These types of devices present a challenge to the CUSS virtual device component model, as it is difficult to define a way of identifying, setting up and using these devices and data formats that is consistent across CUSS platforms and applications.

6.1 Practical and Technical Considerations

Here are the main issues to address in defining how these types of device are used in the CUSS device component model:

1. The number of distinct data types supported by a device might be quite large. For example, an ICAO-compliant RFID passport reader can support up to twenty different RFID data formats, in addition to all other optical or OCR formats. A magnetic card encoder can support numerous different encoding formats (for hotel door locks, for example.)
2. The hardware device may or may not support reading all the data at once (vs. selectively reading only an individual data element.)
3. Reading or writing some data types may be very expensive operations. For example, some document scanner operations can take in excess of 20 second to perform a full scan and document validation.
4. A component model with extended device/data support should not be implemented in any way that may mislead a CUSS application which has no knowledge or logic for handling

these extended data types. For example, if an application only wants a normal passport reader component, it should not "accidently" get an RFID passport scanner component.

5. Even though the component model supports multiple linked virtual components, the level of linking must remain relatively simple, for complexity issues both of platform implementation and application design and detection.
6. Many applications require the ability to read or write different types of data using separate calls to the CUSS platform. This requirement can be the result of business logic within the application, bandwidth constraints between the kiosk and the processing system, legal requirements that prevents applications from reading some data unless certain conditions are met.

6.2 Identifying an Extended Data Component

Each specific type of extended data is identified by a unique string starting with "DS_TYPES" and an associated unique number.

The string is used as part of a component device characteristic, and lets the CUSS application identify and use the correct component for that type. The number is used as an index to insert or extra data into the component data events and requests.

The full list of extended DS_TYPES is presented in Appendix H. This list may be amended from time to time in order to add new data types, as CUSS kiosks integrate new devices. Please consult any updated Addendum documents for more details.

A platform must implement and an application must support the following process for extended data types, if they need support for the non-standard data type in question:

1. Examine the Appendix to determine which data types are required and suitable for the device being used.
2. For each MediaInput and/or MediaOutput virtual component for that device, include in the firmwareVersion Characteristics value all the strings (zero, one, or more) for the extended types of data supported by the component. Different components may support different types (see below.)
3. To identify support to a specific data type, include the string literal of the constant used for that data type, such as "DS_TYPES_VING" for *const long DS_TYPES_VING = 1000*;
4. An application should examine the firmwareVersion and other component characteristics it needs to locate and use the devices and components it needs.
5. The default media type DS_TYPES_ISO is assumed and does not need to be explicitly listed within firmwareVersion.
6. If multiple DS_TYPES are included in a single component, they must be space or comma-delimited.

7. CUSS applications should only use these DS_TYPES indication characteristics, and not on other inappropriate fields such as the RealComponentName.

Here are sample characteristics firmwareVersion strings defining data support:

```
7654327V1.17(h07/31/03) (DS_TYPES_IMAGE_IR,DS_TYPES_IMAGE_UV,DS_TYPES_BARCODE)
DS_TYPES_ISO DS_TYPES_VING DS_TYPES_SAFLOK
```

6.2.1 Setting up and Using an Extended Data Component

To maintain behaviour consistent with the CUSS standard by default:

1. A CUSS platform must not enable any extended media type unless specifically requested by an application via the setup() directive.
2. The platform resets the media type after each application session. In other words, *the media type setup is reset after each session and is not context-persistent.*
3. The application must call setup() when it is ACTIVE, prior to calling enable(). The platform shall ignore calls to setup() while the device is enabled, due to hardware restrictions that may exist on enabled devices.
4. If an application calls setup() multiple times, each call completely resets the selected media types to only those indicated in the most recent setup() call.
5. It is recommended, but not required, that platforms provide support for extended data types in Single-App Mode with Media Off Roller. In this case, point (2.) would not apply, and applications are obliged to also call setup() in INITIALIZE mode, which settings the platform would then retain and enable while the single application is AVAILABLE.

The application must use the setup() directive to enable extended media types, as described above, both for MediaInput and MediaOutput devices as appropriate.

1. The ds (types::datastream) parameter must be of type msgDataType.
2. The msgDataType parameter must include one record for each media type requested, *including DS_TYPES_ISO if needed!*
3. The dataStatus status field in each record must contain the constant value for the media type that the application wants to enable.
4. The bytestream message field in each record must contain any media type-specific setup information required by that device, if any is listed in the media type table below. Typically this will not be needed, however, but this is made available for future expansion.
5. Each media type should only be listed once within msgDataType.
6. The platform will return RC_UNAUTHORIZED if any of the media types is not supported, and not enable any of the other listed media types.
7. Some media types might have large data objects, or take extra time to physically enable or retrieve on the device. For this reason, apps should only enable the extended types they need, especially if event data is sent “over the wire”.

8. When referring to an image type, the specific type implied is the image format specified within the CUSS 1.2 table of supported technologies.

6.3 Sending and Receiving Extended Data

6.3.1 Obtaining data from an extended MediaInput component

Once the device is enabled for particular type(s) of data, the procedure for receiving data from that device is the same as for normal media input devices (e.g. receive().) However, the msgDataType should contain addition records for the new media types returned by the device.

1. The dataStatus status field for each record will be the sum of the extended media type, and the actual data status for that media record. For example, corrupted VING keylock data would be indicated by data status DS_CORRUPTED+DS_TYPES_VING (1001.) For example, an empty FOID card data track would be DS_ZEROLENGTH+DS_TYPES_FOID (103.)
2. For backwards compatibility, the first records indicate ISO tracks 1, 2 and 3.
3. The bytestream message record for an extended data type is data in the format expected for that type, either as documented in the table below, or as widely understood in the industry.
4. Data records for some types might be very large, especially for image/biometric types.
5. When referring to an image type, the specific type implied is the image format specified within the CUSS 1.0 table of supported technologies.
6. If DS_CORRUPTED is provided by the platform as the data status, then there will be no associated data returned: ie, the platform will not provide the partial/corrupted data to the application.

6.3.2 Sending data to an extended MediaOutput component

When using a MediaOutput device with extended media types, the concept is analogous to the msgDataType described above. However, the application must use the send() directive with a properly-formed msgDataType.

1. The dataStatus status field for each record must be the extended media type, such as DS_TYPES_VING (1000)
2. The bytestream message for sending the extended type is data in the format expected for that type, either as documented in the table below, or as widely understood in the industry.

6.3.3 Support for Validated Data

Certain devices may have the ability to flag aspects of the data they provide as "validated". For example, a secure passport reader may be able to validate that the UV image for a given passport is in fact "valid" based on the expected UV appearance of documents from that passport's issuing country.

1. Any virtual MediaInput or MediaOutput component that supports validation must include "VALIDATE" in the firmwareVersion characteristics string for that component.
2. An application that wishes to receive validated data must include the term "VALIDATE" in the setup() directive in setting up the component.
3. If set up for validation, data received from a component will include a status indicator for the validated or invalid data. See the table below.

6.3.3.1 Validated Data Status Indicators

To support proper status indication for data received from components that support validation and for which validation is enable, data records may include the following new data status (DS) values.

1. DS_DOCUMENT_AUTHENTICATION_FAILED – The document is readable and valid, but modifications or tampering has been detected.
2. DS_MISMATCH – The document is readable and valid, but the data from one aspect of the document (for example, RFID) does not match a different aspect (for example, the MRZ)
3. DS_INVALID – The document is readable but considered invalid (as determined by the device validation mechanism.)

6.3.4 Component Model for Extended Devices

The CUSS virtual component linking model provides a very flexible way of exposing to CUSS applications the features of a real device. To allow proper and consistent operation of the CUSS platform and CUSS applications, and to minimize the complexity of implementing and using these extended devices, the following are agreed as being design points for setting up extended data component linking:

1. The data model can NEVER result in a case where an application might incorrectly detect the wrong component when it is following the Real Device Behaviour Clarification guidelines. For example, a component with extended data types cannot "look" like a

normal passport reader, etc, if an application is only looking at the standard characteristics and is not "aware" of the DS_TYPES logic.

2. As such, virtual components (MediaInput, MediaOutput) that list DS_TYPES_ in there characteristics must us "nonApplicableMediaType" as their media type characteristics, when needed to avoid false detection.

3. A device that supports multiple extended data types should present a separate virtual component for each "media type" and "group of extended data types", including the standard types. For example, a flatbed scanner with e-passport RFID support would be four components:

    a. "normal" passport MediaInput (Printed, notApplicableBarcodeStd) for OCR data
    b. "normal" barcode MediaInput (Printed, usedBarcodeStd128) for scan data
    c. "special" image MediaInput (nonApplMediaType) listing all DS_TYPES_IMAGE types it supports
    d. "special" e-chip MediaInput (nonApplMediaType) listing all DS_TYPES_EPASSPORT_DGx types it supports.

4. For example, a scanner that supports **only** 2D PDF417 scanner should be a MediaInput, on a nonApplicableMediaType, with a notApplicableBarcodeStandard, and listing DS_TYPES_SCAN_PDF417 in characteristics. That way only an app that is explicitly looking for that extended type will find/use this component.

5. On the other hand a typical CUSS 1.2-compliant scanner that supports normal 1D barcodes in addition to PDF417 (and possibly other 2D barcode types) should use a single BarcodeStandard::Code128 component instead with the MediaTypeDef set to Printed as listed in Chapter 7, as false detection is not an issue.

6. A passport scanner that supports both 1D and 2D barcodes in compliance with CUSS 1.2 is *not* required to report in its Characteristcs which types of 2D barcode are supported. Application this must not assume that, for example, DS_TYPES_SCAN_PDF417 will be reported on all barcode scanners, as PDF417 support is a requirements for CUSS 1.2 compliance.

7. On the other hand a scanner that supports normal 1D barcodes in addition to PDF417 could us a single usedBarcodeStd128 component instead, as false detection is not an issue.

8. Applications should ignore the "media type" characteristics if it is searching for a specific component that has this media type. This is a requirement since the type might now be "nonApplicable" instead of "Chip" or "Printed".

9. Legacy CUSS 1.0 and CUSS 1.1 implementations based on the DS_TYPES approaches covered in A.1.34 and A.1.45 are not required to change for CUSS 1.2 compliance, due to the limit scope of their deployment on proprietary CUSS hotel and airline kiosks.

10. New devices including e-Passport scanners must follow this revised model to be CUSS 1.2 compliant.

6.4 Non-AEA Printing on General Purpose Printers (GPP)

This section discusses how kiosk applications can use General Purpose Printers (GPPs) that are available on CUSS kiosks, to product more complex documents than allowed by the AEA print standard.

As mentioned in Section 4.1.1, the minimum printer requirement on a CUSS kiosk is to have a Boarding Pass printer supporting the AEA printer language. Some kiosks, typically those that use a legacy ATB2-style printer with native AEA support, do not support GPP printing or non-AEA print data. For this reason, even if a CUSS application can use GPP printing, it should always have the ability to "fall back" to AEA printing only, on kiosks that do not support GPPs. For more detail on printing with AEA, please read Appendix D and Chapter 7.

> **Important Note:** a CUSS kiosk may provide more than one GPP printer. For example, a kiosk may include a receipt printer, a boarding pass printer, and a specialty printer such as a Heavy Tag printer for bag drop kiosks.
>
> CUSS applications that use SVG should not assume all GPP printers are of equal capability. For example, applications with receipt printing functions should attempt to use the printer that provides a PrintSize that matches the applications's requirements for receipts, instead of the first available GPP printer.

6.4.1 Printing using SVG (Scalable Vector Graphics)

New applications written for CUSS 1.2 that use a GPP may prefer to implement PDF documents (see Section 6.4.2 below) instead of SVG, as PDF is more suitable to documents.

A CUSS 1.2 kiosk that provides a General Purpose Printer (GPP) that lists SVG in its MediaOutput supportedDataTypes characteristic must support SVG printing. For more information on finding and using a GPP, please review Section 7.11.

To properly print SVG documents, applications shall follow these requirements. Compatibility problems with SVG printing should be brought to the attention of the CUSS Technical Solution Group for resolution.

1. Examine the maxPrintSizeX and maxPrintSizeY printer characteristics (in millimetres) to create documents that will fix correctly on the page. If larger documents are sent to the GPP the result may be scaled or truncated and not appear correctly. Platforms must ensure that these characteristics values are accurate (not to 0mm.)

2. If the SVG print data refers to external resources (such as image files) provided by the application, the SVG data stream must include the full path and filename of the resource. Applications should use the path returned from the Storage component to detect where the application root directory on the kiosk is.

3. Applications shall generate SVG print data that includes to full name space inside the <svg> block: xmlns="http://www.w3.org/2000/svg" xmlns:xlink=*http://www.w3.org/1999/xlink*

4. SVG document size shall be limited to at most 12MB.

5. Applications must embed any non-standard resources inside the SVG as needed. The application can assume that the default SVG fonts (including Unicode/multi-byte) are available on the kiosk. But, for example, barcode fonts used by the application must be embedded inside the SVG.

6. Any embedded references inside the SVG data that link to external files such as image or DTD files must be resolvable and accessible from the kiosk, for example either as local/server files in the Storage component directory, or from an accessible application web server URL. For example, in "<!DOCTYPE svg SYSTEM '*C:\APPS\Storage\ZZ\SVG\svg10.dtd*' >" the .dtd file is readable from the kiosk.

6.4.2 Printing using Adobe PDF (Portable Document Format)

In CUSS 1.2, any kiosk that includes a General Purpose Printer (GPP) which supports SVG printing must also support PDF printing. Or, speaking in terms of component Characteristics, if a CUSS 1.2 platform provides a General Purpose Printer (GPP) that lists DataType::SVG in its MediaOutput supportedDataTypes characteristic, then that MediaOutput component must also support PDF printing. For more information on finding and using a GPP, please review Section 7.11. GPP printers must be at least 200dpi.

For backwards compatibility, the CUSS platform must indicate "DS_TYPES_PRINT_PDF" in the firmwareVersion characteristic of its general purpose printer.

To properly print PDF documents, applications should follow these requirements. Compatibility problems with PDF printing should be brought to the attention of the CUSS Technical Solution Group for resolution.

1. Examine the maxPrintSizeX and maxPrintSizeY printer characteristics (in millimetres) to create documents that will fix correctly on the page. If larger documents are sent to the GPP the result may be scaled or truncated and not appear correctly. Platforms must ensure that these characteristics values are accurate (not to 0mm.)

2. If the SVG print data refers to external resources (such as image files) provided by the application, the SVG data stream must include the full path and filename of the resource. Applications should use the path returned from the Storage component to detect where the application root directory on the kiosk is.

3. Multiple-page PDF documents are allowed, with no limit on the total number of pages sent. Applications should use the printer Bin characteristics to determine the capacity of the printer.

4. PDF document size shall be limited to at most 12MB. Applications should use the inline compression features of PDF to reduce stream size and overhead.

5. Applications must embed any non-standard resources inside the PDF as needed. The application can assume that the default PDF fonts (including Unicode/multi-byte) are available on the kiosk. But, for example, barcode fonts used by the application must be embedded inside the PDF.

6. Applications cannot use or depend on any proprietary extensions to the PDF standard.

7. The application must send PDF print data to the send() and setup() commands. The platform must parse the request data to determine if the data is SVG or PDF and process it accordingly.

If a platform detects that it cannot parse or print the PDF document, it will return FORMAT_ERROR or other suitable code to reflect the error condition as a result of the send() request.

6.4.3 Reverse/2-sided printing on GPPs

Some kiosks vendors may choose to include in their kiosks printer hardware that supports impression on both sides of the paper. The platform can use this feature directly (for example, to print advertising or information on the reverse of all kiosk documents, without any input from the kiosk applications) and can also extend this capability to CUSS applications running on the platform. If a platform/kiosk vendor provides this hardware capability to applications running on the kiosk, the platforms should offer and applications can detect and use (if desired) 2-sided printing on a CUSS kiosk using the following methodology.

The only data format available for 2-sided printing is PDF. Therefore, a platform can support 2-sided printing only if it supports PDF printing as discussed in Section 6.4.2. All recommendations for PDF printing apply to PDF documents provided for 2-sided printing.

1. Except where noted below, “TWOSIDED” must appear in the firmwareVersion of the Manufacturer characteristics of the MediaOutput component of a printer which supports 2 sided printing.

2. 2-sided printers that are configured at the platform or hardware level to print specific information on the back must not report the "TWOSIDED" characteristic. From an application perspective, such hardware appears to be and behaves as a single-sided GPP.

3. A platform configured with a 2-sided printer hardware but operating as a normal single-side GPP printer must not accept any 2-sided printing commands as listed below (ie an application must not be able to interfere with any platform/hardware 2-sided operation if 2-sided printing is not advertised to the application.)

4. The reverse side of a 2-sided document shall be oriented by default such that when the document is flipped along its vertical edge, the content on the reverse side is right-side-up. In other words, side binding (like a book) is used by default, not top binding.

5. GPP support for back-side printing a single page is indicated by the presence of DS_TYPES_PRINT_2S_PAGE.

    a. setup() is used to send the data to print on the back side using svgDataType[]
    b. setup() accepts a datastream with a PDF document to print on the back side of the document.
    c. The datastream input will be prefixed with ~~2S_PAGE~~ to indicate that the PDF data is to be printed on the back side.
    d. The PDF document specified remains in the context for all subsequent print requests from that application until it is replaced with another PDF document or cleared
    e. To clear the PDF document, the datastream input will be ~~2S_PAGE~~ with no data after the text.
    f. The PDF document specified for the back side must be a single page PDF. If a multi page PDF is provided as part of the setup() call, then the platform shall return FORMAT_ERROR.
    g. If a platform detects that it cannot parse or print the PDF document, it will return FORMAT_ERROR or other suitable code to reflect the error condition as a result of the setup() request.
    h. ~~2S_PAGE~~ printing is cleared when the printer is released. Back-side documents set by an application must only print for print requests from that application (the back-side printing information is part of the platform's application context and must not persist to other applications.)

6. GPP support for front-back printing of multi-page documents is indicated by the presence of DS_TYPES_PRINT_2S_MULTI.

    a. setup() is called to request 2-sided printing of a sequence of pages using svgDataType[]
    b. datastream is ~~2S_MULTI_ON~~
    c. ~~2S_MULTI_ON~~ is mutually exclusive with ~~2S_PAGE~~. That is, each request implicitly cancels the other.

d. Multi-page front-back printing is supported through the use of multi-page PDF documents. Each new document starts on the next front page. That is, it is not possible to send subsequent single-page PDF documents using ~~2S_MULTI_ON~~ and have every other PDF document print on the back. If such functionality is required, it is preferred to use ~~2S_PAGE~~ as described above.
e. datastream to return to single-sided printing is ~~2S_MULTI_OFF~~
f. If an application prints multi-page PDF (Section 6.4.2) without specifying ~~2S_MULTI_ON~~, then the platform prints normal 1-sided documents.

6.4.4 Page margins and printable area

If a kiosk platform uses a General Purpose Printer which has a physical printable area that is smaller than the logical print area, it must advise the application of this limitation. An example of this would be where the printer or platform chops/truncates print data sent to the far (0,0) corner because of a printer hardware or platform software limitation.

Ideally, the entire print data is properly printed on GPPs with no truncation. If this is not possible, however, the platform must implement the following behaviour.

1. The maxPrintSizeX and maxPrintSizeY characteristic values must be properly set by the platform, in millimetre (see Section 5.10.)

2. The Manufacturer::firmwareVersion characteristic of the GPP MediaOutput must include the “originPointX=<value>” field if the printer truncates data horizontally (the left side of the print data does not appear.) This value is in mm.

3. The Manufacturer::firmwareVersion characteristic of the GPP MediaOutput must include the “originPointY=<value>” field if the printer truncates data vertically (the top of the print data does not appear.) This value is in mm.

4. These settings apply to GPP printing with SVG, PDF, or 2-sided PDF.

5. All standard AEA printing MUST print properly on any platform/kiosk printer that indicates support for AEA, without truncation. Ie, any platform printer component that supports aeaDataType must print valid AEA request without truncation and in accordance with the ATB document specification, even if it via GPP printer.

The applications can use these setting to adjust and position their SVG or PDF print data to print correctly on the kiosk.

6.4.5 Receipt Printing and Specialty Document Printing

Many kiosks applications may need to print documents other than ATB-format boarding documents. This can include itineraries, payment or baggage receipts, or other specialty items such as Heavy Tags for self baggage drop kiosks.

In all cases, recall that a CUSS is only *required* to offer an ATB boarding pass printer supporting the AEA language. All other types of printers are optional. So a well-written application should be able to detect and cope with a situation where a kiosk does not include the exact type of printer it needs.

For example, an application should not require GPP printing support (for receipts or any other document) and expect to operate on all CUSS kiosks worldwide, as there are numerous kiosks that cannot or choose not to implement GPP printing.

As a general rule, CUSS applications should probably follow an approach such as this to detect and use the correct printer(s) it needs.

1. Examing the CUSS component list and identify all AEA and GPP printers available to the application.

2. Review the characteristics of each printer to classify their capabilities:
    a. Is it AEA, or GPP with SVG/PDF support?
    b. What size document does it support (see 6.4.4)
    c. Is the document Portrait or Landscape
    d. Does the Manufacturer.firmwareVersion characteristic indicate any special printer media support, such as DS_TYPES_HEAVYTAG?

3. Acquire and use the printers that are most appropriate for the capabilities that exist in the application.

4. If an expected printer is not found, either set the application to the UNAVAILABLE state, or revert if possible to an alternate printing logic that produces documents using the mandatory AEA boarding pass printer component on the kiosk instead of the GPP.

Also note that in many cases, printing of specialty documents such as Receipts or Heavy Tags, may include specific document content and formatting requirements that are not specified by the CUSS standard but are instead imposed by airport or other industry requirements. For example:

- Payment receipts may need to include specific data and formatting mandated by the acquiring bank
- Heavy Tags may have a different stock type or size depending on the airport or country

Finally, note that in some cases, additional printing functionality might exist on specialty kiosks. For example, self bag drop kiosks may offer receipt printing support via AEA-SBD protocol commands in addition to a CUSS GPP printer interface.

Ch 7: Real Device Programming Guide

This chapter is based on the CUSS 1.1 document "*Clarification of IATA CUSS Real Device to Virtual Component Mapping"* with some formatting and layout changes.

This chapter is a programming reference that clarified the real device behavior when the devices are used on the IATA CUSS v1.2 implementation. It documents the virtual component linkage and characteristics to allow a CUSS application developer to better understand the mapping between a real device and a set of virtual components and characteristics for that device.

To simplify the sequence diagrams, assume that all components are already acquired via the acquire() call and release via release() will be called if the application is finished using / listening to the virtual components.

The sequence diagrams are not intended to show every possible scenario. They are illustrations of typical usage cases, as well as some cases that may be harder to conceptualize. In theory, error scenarios (such as device becoming not responsive) can occur at any time, independent of the operation being performed.

7.1 List of Figures

7.2 Simple ATB Printer (AEA Printing Device)

Description of Device:

The Simple ATB Printer is a simple AEA printing device without magnetic encoding capability and no insertion slot to read / revalidate coupons. It does not have an escrow device and once the coupons are printed, they are presented to the end user automatically.

Virtual Component Linking Diagram:

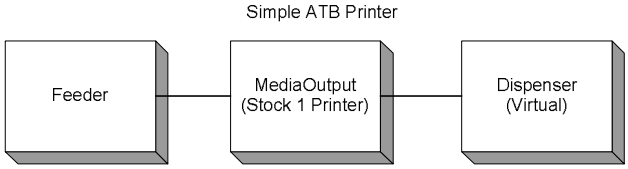


Figure 1: Linking for Simple AEA printing device

Description of Virtual Component Linkage:
A MediaOutput component is linked to a Feeder and a virtual Dispenser

Distinct Characteristics:

MediaOutput (Stock 1 Printer)	
Characteristic	**Value**
MediaType.MediaTypeListDef	MediaType.MediaTypeDef.Printed
MediaOutput.MediaType.type	MediaOutput.MediaType.BoardingPass MediaOutput.MediaType.Ticket MediaOutput.MediaType.GeneralPurposeDoc
MediaOutput.supportedDataTypes	DataType.AEA

Dispenser (Stock 1 Printer)	
Characteristic	**Value**
Dispenser.kind	Dispenser.DispenserType.virtual_ The dispenser is virtual because an offer() is not required. This is because once printed, the media is available to the end user. If offer() is called the platform will generate the response with status code OK so that application knows that the platform does not have a physical sensor in the dispenser component. If offer() is called against the virtual dispenser, the offer() must block until the tickets are removed if the platform supports the ability to detect removal of coupons.

Typical Sequence Diagram for AEA Printing Device for Printing:

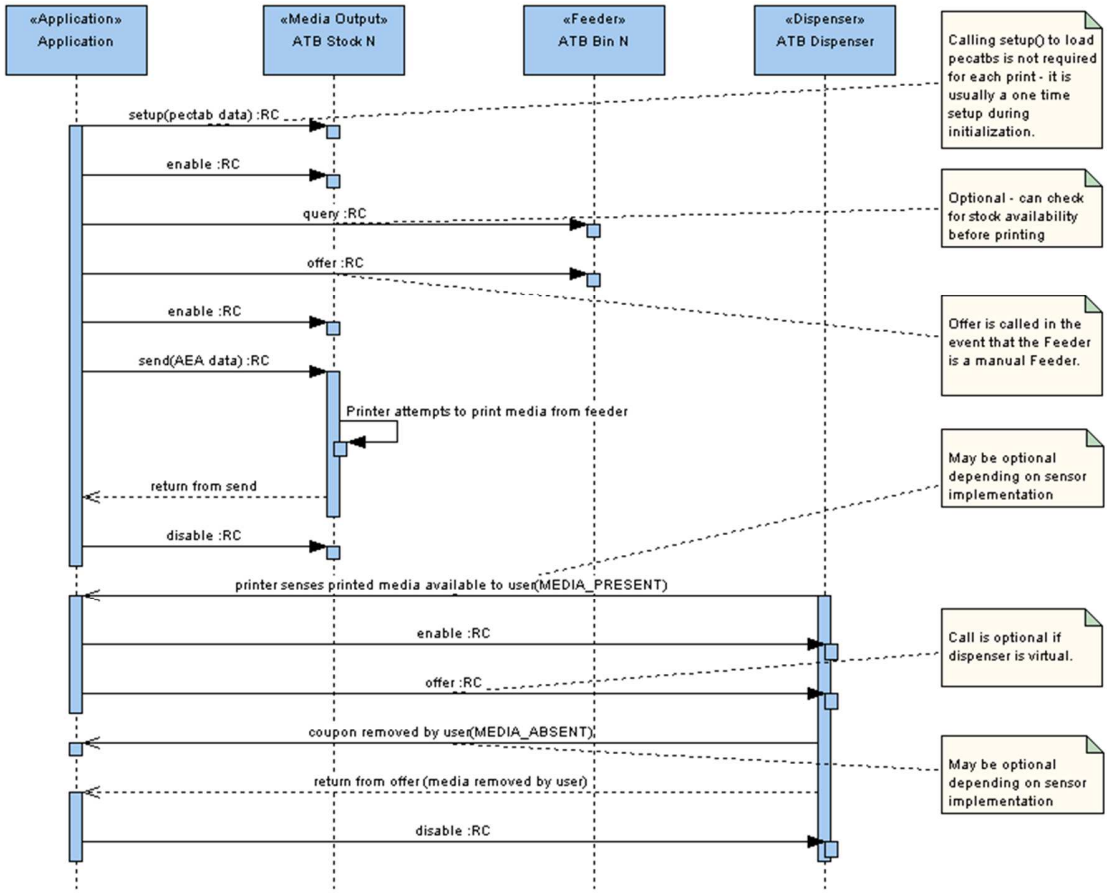


Figure 2: Printing AEA Coupons

7.3 ATB/2 with Insertion Slot

Description of Device:

The ATB/2 with Insertion Slot Printer has magnetic encoding capability and the ability to read / revalidate coupons inserted by the end user. It does not have an escrow device and once the coupons are printed, they are presented to the end user automatically. In this example, coupons inserted into insertion slot that become revalidated or ejected are sent to the main coupon tray. The printer may have a number of bins.

Virtual Component Linking Diagram:

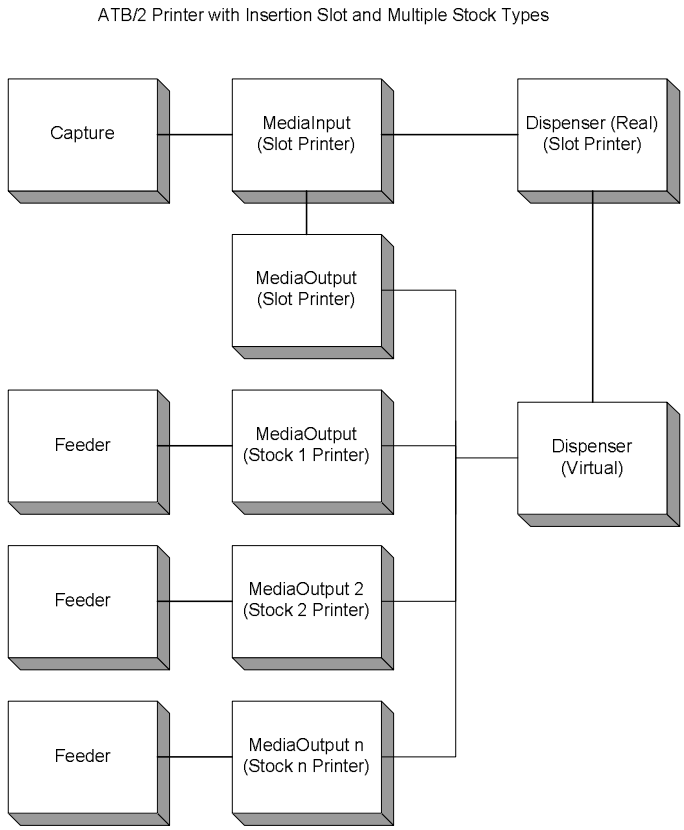


Figure 3: Linking for ATB Printer with insertion slot and multiple bins

Description of Virtual Component Linkage:
There is a MediaOutput component defined for each Feeder. Each MediaOutput component can have the same media type, or a different media type, depending on configuration. There is also a MediaOutput and a MediaInput component associated with the Insertion Slot. The MediaOutput of the Insertion Slot and the MediaOutput of the Feeder components are linked to a virtual dispenser. Since there is no escrow, media that is printed can be retrieved directly by the end user. The MediaInput of the Insertion Slot is linked to a real Dispenser since offer() is required for the end user to retrieve coupons that are inserted but not revalidated. The real Dispenser can be linked to the virtual Dispenser if coupons that are ejected exit the printer via the main tray.

Distinct Characteristics:

MediaOutput (Stock 1, 2, … , n Printer)	
Characteristic	**Value**
MediaType.MediaTypeListDef	MediaType.MediaTypeDef.Printed MediaType.MediaTypeDef.MagneticStripe
MediaOutput.MediaType.type	MediaOutput.MediaType.BoardingPass MediaOutput.MediaType.Ticket MediaOutput.MediaType.GeneralPurposeDoc
MediaOutput.supportedDataTypes	DataType.AEA
MediaOutput.numberOfTracks	1

MediaOutput (Slot Printer)	
Characteristic	**Value**
MediaType.MediaTypeListDef	MediaType.MediaTypeDef.Printed MediaType.MediaTypeDef.MagneticStripe
MediaOutput.MediaType.type	MediaOutput.MediaType.InsertedDoc
MediaOutput.supportedDataTypes	DataType.AEA
MediaOutput.numberOfTracks	1 In AEA, the data is encoded on 4 tracks but the track number is hidden from the application.

MediaInput (Slot Printer)	
Characteristic	**Value**
MediaType.MediaTypeListDef	MediaType.MediaTypeDef.Printed MediaType.MediaTypeDef.MagneticStripe
MediaInput.typeOfReader	MediaInput.ReaderType.Motorized
MediaOutput.supportedDataTypes	DataType.AEA

Dispenser (Main Tray)	
Characteristic	**Value**
Dispenser.kind	Dispenser.DispenserType.virtual_ The dispenser is virtual because an offer() is not required. This is because once printed, the media is available to the end user. If offer() is called the platform will generate the response with status code OK so that application knows that the platform does not have a physical sensor in the dispenser component. If offer() is called against the virtual dispenser, the offer() must block until the tickets are removed if the platform supports the ability to detect removal of coupons.

Dispenser (Insertion Slot)	
Characteristic	**Value**
Dispenser.kind	Dispenser.DispenserType.real_ The dispenser is real because an offer() is required to eject the inserted coupon. This is because once inserted, the media is only available to the end user after it has been ejected. In this example, the dispenser of the insertion slot is linked to the virtual dispenser of the main tray because in this example it is assumed that coupons eject from the main tray instead of the insertion slot.

	** In CUSS 2.0, this dispenser will be a Feeder virtual component. Further definition is required to determine the characteristics of this new feeder virtual component **

Capture	
Characteristic	**Value**
Bin.BinSize	Maximum number of documents a bin can hold
Bin.AlmostFullLevel	Shows the high threshold of the bin if corresponding sensor is installed. May generate a MEDIA_HIGH event.
Bin.currentNoOfDocuments	Shows the current number of documents in the bin

Typical Sequence Diagram for ATB/2 with Insertion Slot (No Escrow):

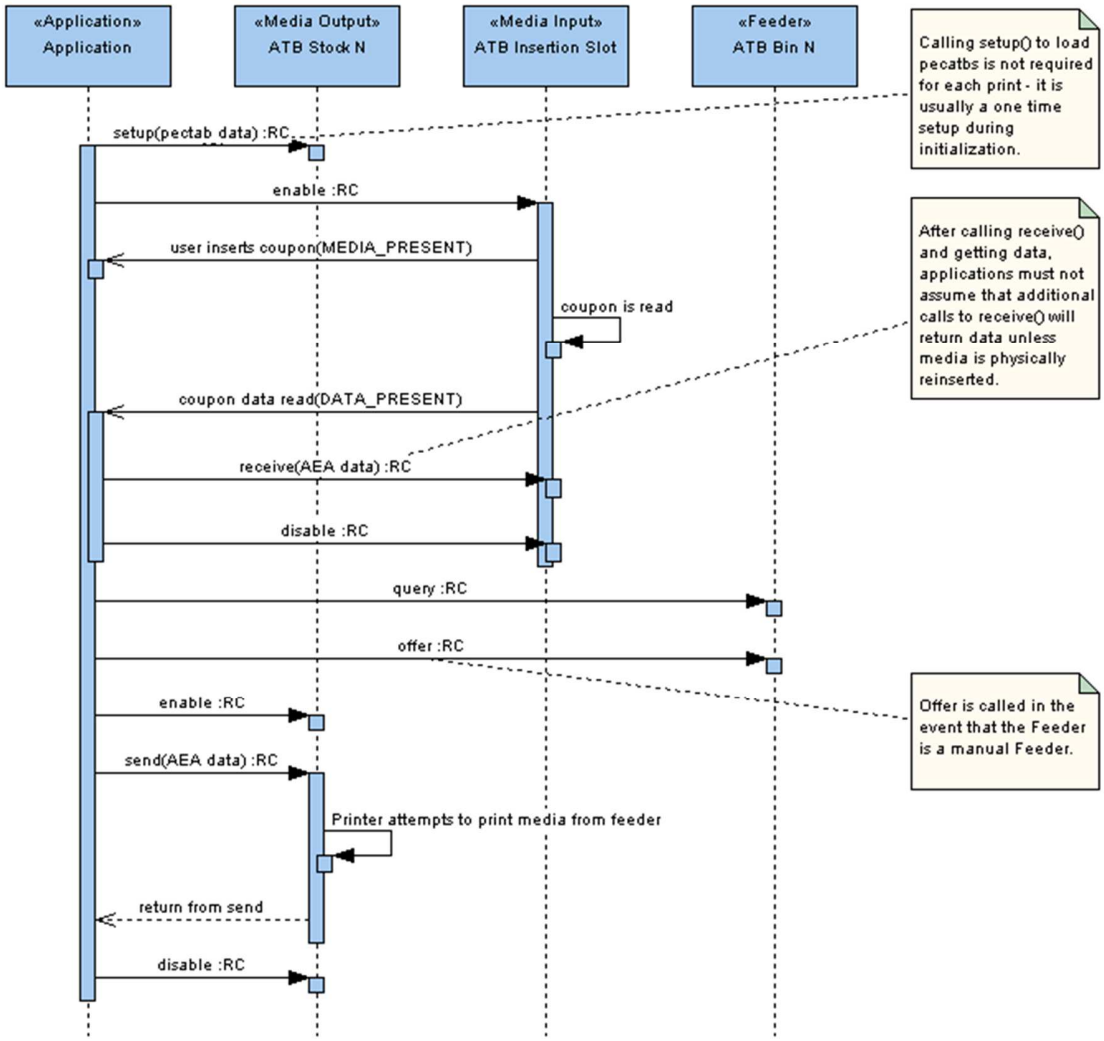


Figure 4: Reading and printing AEA Coupons (no Escrow device attached)

7.4 ATB/2 with Insertion Slot and Escrow

Description of Device:

The ATB/2 with Insertion Slot Printer has magnetic encoding capability and the ability to read / revalidate coupons inserted by the end user. It has an escrow – a device to hold tickets after they are printed or ejected but before they are offered to the user. If there is a problem during printing of some coupons, the capture virtual component connected to the escrow represented as a dispenser can be used to retain the current contents inside the escrow. In this example, coupons inserted into insertion slot that become revalidated or ejected are sent to the main coupon tray. The printer may have a number of bins. There are two Capture virtual components – the Capture component associated with the MediaInput can be used to capture inserted coupons, while the Capture component associated with the escrow can be used to capture that is currently inside the escrow device.

Virtual Component Linking Diagram:

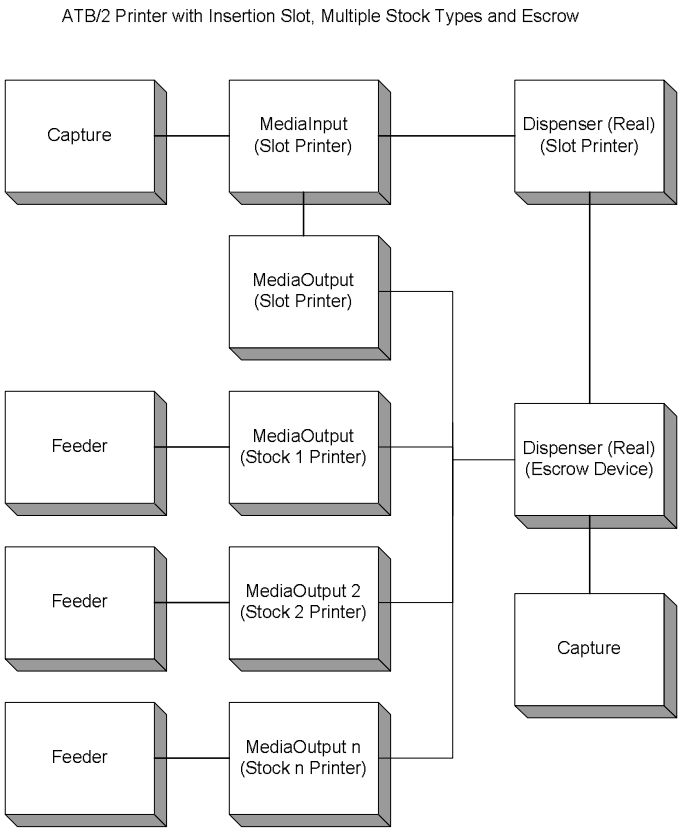


Figure 5: Linking for ATB Printer with insertion slot, multiple bins and escrow

Description of Virtual Component Linkage:

There is a MediaOutput component defined for each Feeder. Each MediaOutput component can have the same media type, or a different media type, depending on configuration. There is also a MediaOutput and a MediaInput component associated with the Insertion Slot. The MediaOutput of the Insertion Slot and the MediaOutput of the Feeder components are linked to an escrow device, defined as a real dispenser. Since there is an escrow, media that is printed can be retrieved by the end user after an offer() call from the real

dispenser. The MediaInput of the Insertion Slot is linked to a real Dispenser since offer() is required for the end user to retrieve coupons that are inserted but not revalidated. The real Dispenser of the insertion slot is linked to the real Dispenser represented by the escrow if coupons that are ejected exit the printer via the escrow. A capture virtual component can be linked to the escrow device if the escrow has the capability to capture coupons. Revalidated tickets from the insertion slot will travel into the escrow.

Distinct Characteristics:

MediaOutput (Stock 1, 2, … , n Printer)	
Characteristic	**Value**
MediaType.MediaTypeListDef	MediaType.MediaTypeDef.Printed MediaType.MediaTypeDef.MagneticStripe
MediaOutput.MediaType.type	MediaOutput.MediaType.BoardingPass MediaOutput.MediaType.Ticket MediaOutput.MediaType.GeneralPurposeDoc
MediaOutput.supportedDataTypes	DataType.AEA
MediaOutput.numberOfTracks	1

MediaOutput (Slot Printer)	
Characteristic	**Value**
MediaType.MediaTypeListDef	MediaType.MediaTypeDef.Printed MediaType.MediaTypeDef.MagneticStripe
MediaOutput.MediaType.type	MediaOutput.MediaType.InsertedDoc
MediaOutput.supportedDataTypes	DataType.AEA
MediaOutput.numberOfTracks	1 In AEA, the data is encoded on 4 tracks but the track number is hidden from the application.

MediaInput (Slot Printer)	
Characteristic	**Value**
MediaType.MediaTypeListDef	MediaType.MediaTypeDef.Printed MediaType.MediaTypeDef.MagneticStripe
MediaInput.typeOfReader	MediaInput.ReaderType.Motorized
MediaOutput.supportedDataTypes	DataType.AEA

Dispenser (Escrow)	
Characteristic	**Value**
Dispenser.kind	Dispenser.DispenserType.real_ The dispenser is real because an offer() required in order for the end user to access media. This is because once printed, the media is inside the escrow device. If offer() is called against the real dispenser representing the escrow, the offer() will block until the tickets are removed from the escrow

Dispenser (Insertion Slot)	
Characteristic	**Value**
Dispenser.kind	Dispenser.DispenserType.real_ The dispenser is real because an offer() is required to eject the inserted coupon. This is because once inserted, the media is only available to the end user after it has been ejected. In this example, the dispenser of the insertion slot is linked to

	the real dispenser of the escrow because in this example it is assumed that coupons eject into the escrow instead of the insertion slot. ** In CUSS 2.0, this dispenser will be a Feeder virtual component. Further definition is required to determine the characteristics of this new feeder virtual component **

Capture	
Characteristic	**Value**
Bin.BinSize	Maximum number of documents a bin can hold
Bin.AlmostFullLevel	Shows the high threshold of the bin if corresponding sensor is installed. May generate a MEDIA_HIGH event.
Bin.currentNoOfDocuments	Shows the current number of documents in the bin

Typical Sequence Diagrams for ATB/2 with Insertion Slot and Escrow:

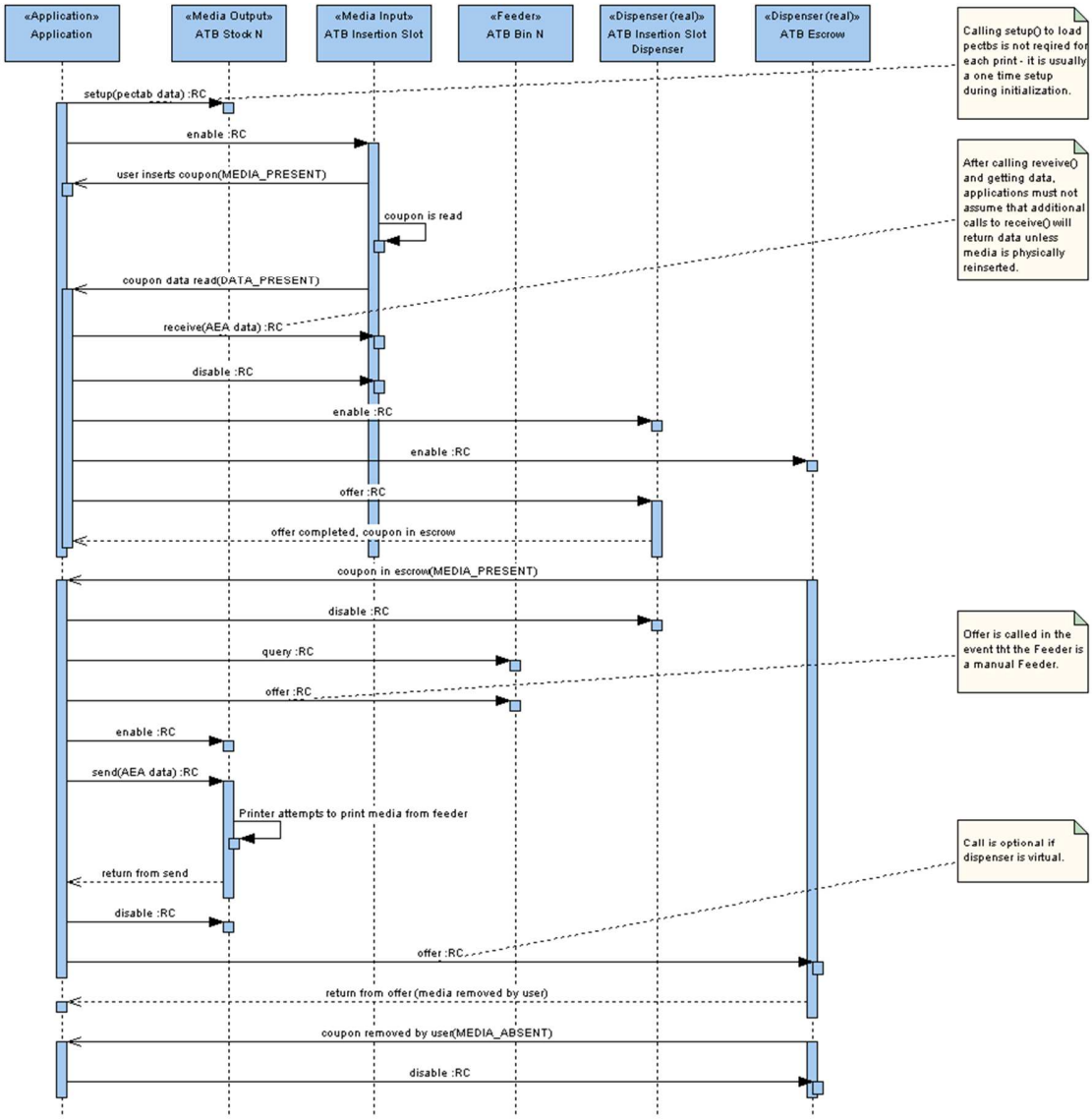


Figure 6: Reading and printing AEA Coupons (with Escrow device attached)

7.5 ATB/2 with Insertion Slot and Escrow (ins. coupons do not eject into escrow)

Description of Device:

The ATB/2 with Insertion Slot Printer has magnetic encoding capability and the ability to read / revalidate coupons inserted by the end user. It has an escrow device to hold tickets after they are printed. However, in this example, coupons inserted into insertion slot that become revalidated is directed to the escrow, while tickets that are not revalidated are ejected out of the insertion slot instead of ejecting into the escrow. The printer may have a number of bins.

Virtual Component Linking Diagram:

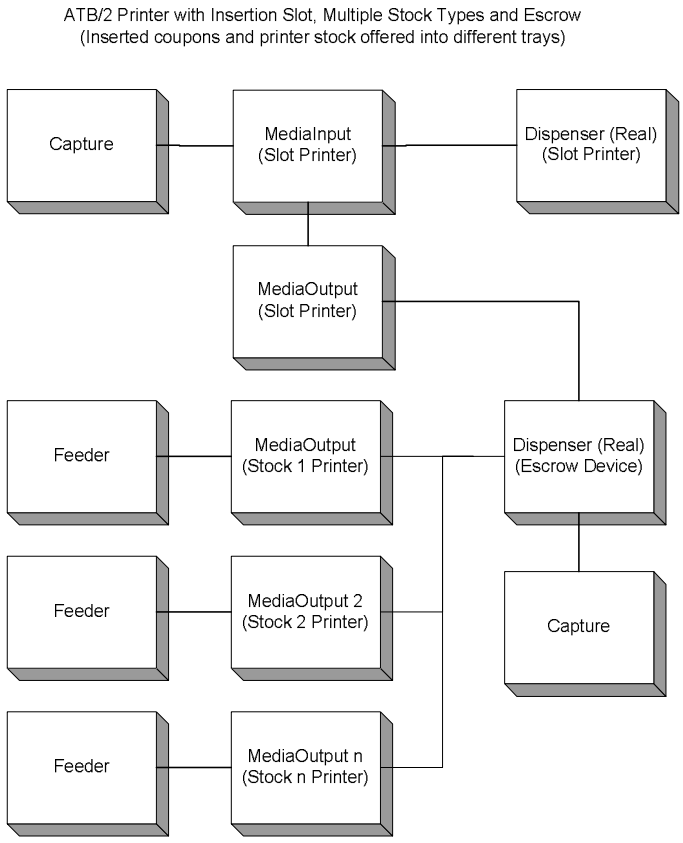


Figure 7: Linking for ATB Printer with insertion slot, multiple bins and escrow (inserted coupons do not eject into escrow)

Description of Virtual Component Linkage:

There is a MediaOutput component defined for each Feeder. Each MediaOutput component can have the same media type, or a different media type, depending on configuration. There is also a MediaOutput and a MediaInput component associated with the Insertion Slot. The MediaOutput of the Insertion Slot and the MediaOutput of the Feeder components are linked to separate real dispensers, one representing the escrow and the other representing the insertion slot. Since there is an escrow, media that is printed can be retrieved

by the end user after an offer() call from the real dispenser. The MediaInput of the Insertion Slot is linked to a real Dispenser since offer() is required for the end user to retrieve coupons that are inserted but not revalidated. The real Dispenser of the insertion slot is not directly linked to the real Dispenser represented by the escrow because coupons inserted from the insertion slot are not always ejected into the escrow (for example, inserted ticket that is not revalidated). A capture virtual component can be linked to the escrow device if the escrow has the capability to capture coupons. Revalidated tickets from the insertion slot will travel into the escrow after calling send() from the MediaOutput virtual component.

Distinct Characteristics:

MediaOutput (Stock 1, 2, … , n Printer)	
Characteristic	**Value**
MediaType.MediaTypeListDef	MediaType.MediaTypeDef.Printed MediaType.MediaTypeDef.MagneticStripe
MediaOutput.MediaType.type	MediaOutput.MediaType.BoardingPass MediaOutput.MediaType.Ticket MediaOutput.MediaType.GeneralPurposeDoc
MediaOutput.supportedDataTypes	DataType.AEA
MediaOutput.numberOfTracks	1

MediaOutput (Slot Printer)	
Characteristic	**Value**
MediaType.MediaTypeListDef	MediaType.MediaTypeDef.Printed MediaType.MediaTypeDef.MagneticStripe
MediaOutput.MediaType.type	MediaOutput.MediaType.InsertedDoc
MediaOutput.supportedDataTypes	DataType.AEA
MediaOutput.numberOfTracks	1 In AEA, the data is encoded on 4 tracks but the track number is hidden from the application.

MediaInput (Slot Printer)	
Characteristic	**Value**
MediaType.MediaTypeListDef	MediaType.MediaTypeDef.Printed MediaType.MediaTypeDef.MagneticStripe
MediaInput.typeOfReader	MediaInput.ReaderType.Motorized
MediaOutput.supportedDataTypes	DataType.AEA

Dispenser (Escrow)	
Characteristic	**Value**
Dispenser.kind	Dispenser.DispenserType.real_ The dispenser is real because an offer() required in order for the end user to access media. This is because once printed, the media is inside the escrow device. If offer() is called against the real dispenser representing the escrow, the offer() will block until the tickets are removed from the escrow

Dispenser (Insertion Slot)	
Characteristic	**Value**
Dispenser.kind	Dispenser.DispenserType.real_ The dispenser is real because an offer() is required to eject the inserted coupon. This is because once inserted, the media is only available to the end user after it has been

	ejected into the escrow instead of the insertion slot. ** In CUSS 2.0, this dispenser will be a Feeder virtual component. Further definition is required to determine the characteristics of this new feeder virtual component **

Capture	
Characteristic	**Value**
Bin.BinSize	Maximum number of documents a bin can hold
Bin.AlmostFullLevel	Shows the high threshold of the bin if corresponding sensor is installed. May generate a MEDIA_HIGH event.
Bin.currentNoOfDocuments	Shows the current number of documents in the bin

7.6 Simple Baggage Tag Printer

Description of Device:

The Baggage Tag printer prints baggage tags according to the BTP AEA specification. In this simple Baggage Tag printer example, the baggage tag is available to the user once it is printed. The baggage tags used should be those with dimensions as stated in the Appendix.

A CUSS kiosk bag tag printer may or may not be able to detect when the kiosk user has taken a bag tag after it is printer. If this capability does exists, a CUSS platform provider may (but is not obliged) to present this capability to CUSS applications by implementing a Dispenser component of type “real”.

It is a CUSS application business logic decision to properly detect and use both types of Dispenser component in accordance their the airline’s internal bag tag printing and/or security requirements.

Virtual Component Linking Diagram:

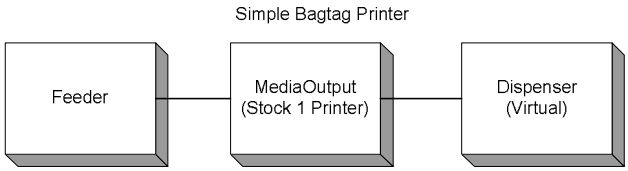


Figure 8: Linking for Simple Baggage Tag Printer

Description of Virtual Component Linkage:

A MediaOutput component is linked to a Feeder and a Dispenser. If the platform monitors if and when bag tags are taken by the user, or there is a physical bag tag output bin, then the Dispenser will be real. Otherwise it is a virtual dispenser.

Distinct Characteristics:

MediaOutput (Stock 1 Printer)	
Characteristic	**Value**
MediaType.MediaTypeListDef	MediaType.MediaTypeDef.Printed
MediaOutput.MediaType.type	MediaOutput.MediaType.BaggageTag
MediaOutput.supportedDataTypes	DataType.AEA

Dispenser (Stock 1 Printer)	
Characteristic	**Value**
Dispenser.kind	**Dispenser.DispenserType.virtual_** If the dispenser is virtual, the platform does not detect when and if a document is taken from the bag tag printer. In this case, an offer() is not required. This is because once printed, the media is available to the end user. If offer() is called against the virtual dispenser, the offer() may block until the tickets are removed if the platform supports the ability to detect removal of baggage tags, even if the Dispenser is type virtual_

	Dispenser.DispenserType.real_ If the dispenser is real, the platform can detect and/or control when the printed document is taken from the bag tag output position. In this case, an offer() is required to make the document available to the end user. Typically, the dispenser for the Baggage Tag printer may only be able to hold a maximum of one baggage tag. As a result, the query of the dispenser component, or asynchronous events may indicate MEDIA_FULL instead of MEDIA_PRESENT The application can use Dispenser.BinSize characteristic to programmatically determine the maximum size of the dispenser bin. In addition, the Dispenser. currentNoOfDocuments can show the number of documents that are in the dispenser. Dispenser.currentNoOfDocuments may not be supported by platforms lacking sensor capability.

Typical Sequence Diagram for Simple Baggage Tag Printer:

The sequence diagram is identical to the Simple AEA printing device. In some cases, the Dispenser virtual component may only have a capacity of one. This is to ensure a baggage tag must be removed from the dispenser prior to another being printed.

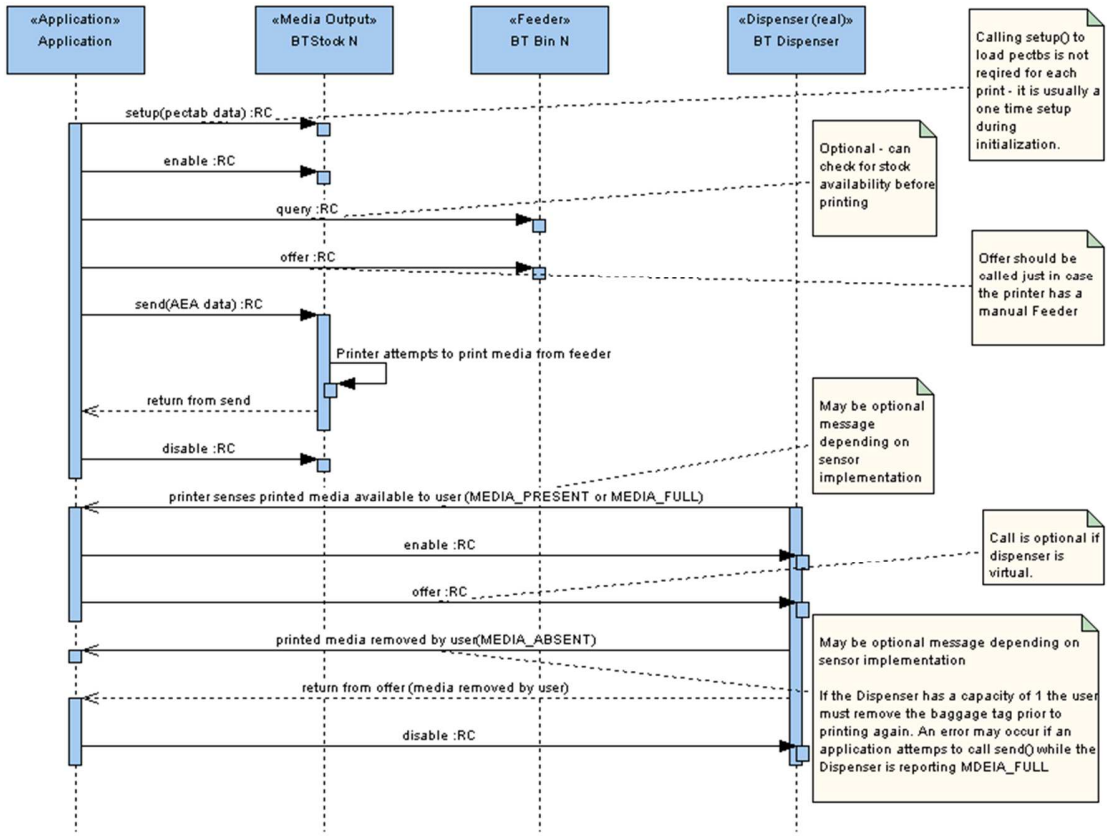


Figure 9: Printing Baggage Tags

7.7 Motorized Magnetic Card Reader

Description of Device:

The motorized magnetic card reader accepts and reads ISO magnetic encoded cards. There is a MediaInput component linked to a real dispenser and a capture component.

Card data for payment cards is provided by the platform in accordance with Chapter 8 (formerly known as the CUSS FOID Addendum.)

Virtual Component Linking Diagram:

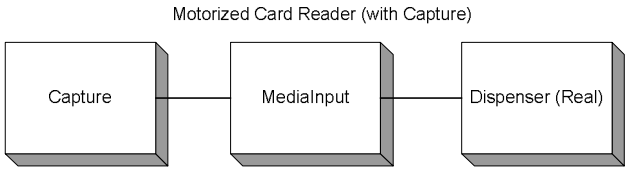


Figure 10: Linking for Motorized Card Reader (with Capture)

Description of Virtual Component Linkage:

The MediaInput virtual component is linked to a real dispenser as the end user can only retrieve an inserted card after the offer() directive is called. The MediaInput virtual component can also be linked to a capture virtual component, used to retain inserted cards.

Distinct Characteristics:

MediaInput	
Characteristic	**Value**
MediaType.MediaTypeListDef	MediaType.MediaTypeDef.MagneticStripe
MediaInput.typeOfReader	MediaInput.ReaderType.Motorized
MediaInput.supportedDataTypes	DataType.MSG
MediaInput.numberOfTracks	<int>, depends on the actual number of tracks the hardware is capable of reading. Typically, this number is 2 or 3 for ISO readers
Manufacturer.FirmwareVersion	<string> - indication of the supported data type, such as: DS_TYPES_FOID_ISO, DS_TYPES_PAYMENT_ISO, DS_TYPES_DISCRETIONARY_ISO. See Chapter 8 for how to use these data types.

Dispenser	
Characteristic	**Value**
Dispenser.kind	Dispenser.DispenserType.real_ The dispenser is real because an offer() is required for the end user to retrieve the card.

Capture	
Characteristic	**Value**
Bin.BinSize	Maximum number of documents a bin can hold

Bin.AlmostFullLevel	Shows the high threshold of the bin if corresponding sensor is installed. May generate a MEDIA_HIGH event.
Bin.currentNoOfDocuments	Shows the current number of documents in the bin

Supported extended data types:

By default, the CUSS card reader interfaces will only provide truncated track data to the application when the customer inserts a payment card. If an application requires full and legitimate access to the payment card information, if must call setup() to configure that access.

Please see the next section 7.8 and Chapter 8 for more information on extended data types and payment card data truncation, as well as a sequence diagram for this usage.

Typical Sequence Diagrams for Motorized Magnetic Card Reader:

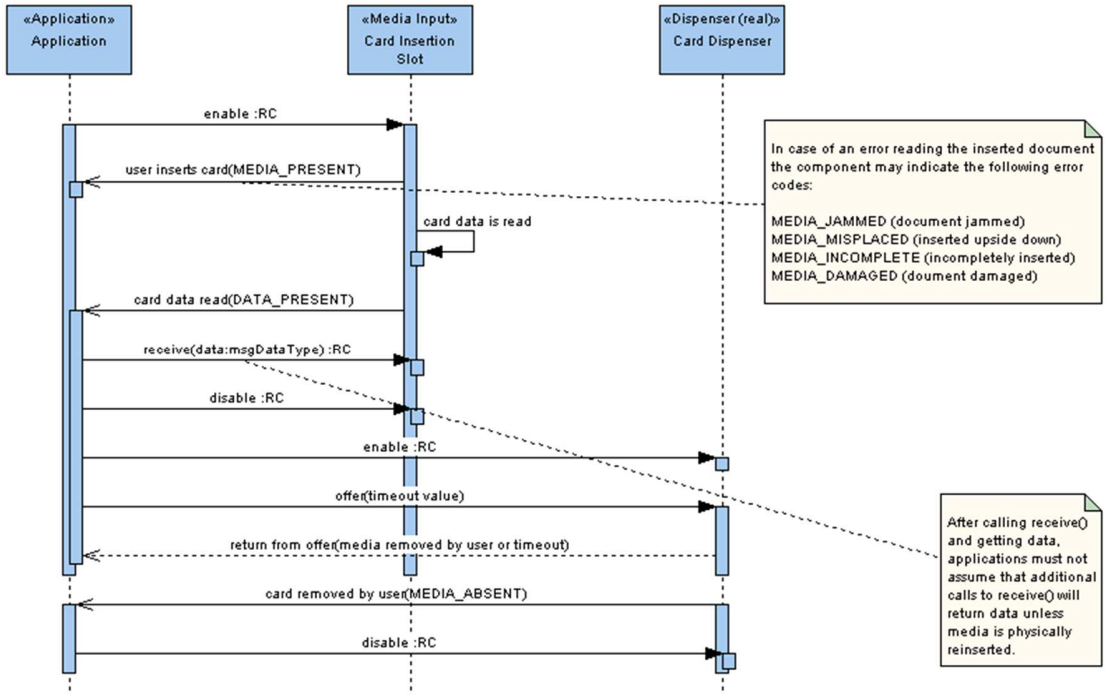


Figure 11: Reading Magnetic Cards on a motorized reader

7.8 DIP / Swipe Magnetic Card Reader

Description of Device:

The DIP / Swipe magnetic card reader accepts and reads ISO magnetic encoded cards. There is a single MediaInput component representing the DIP / Swipe Reader.

Card data for payment cards is provided by the platform in accordance with Chapter 8 (formerly known as the CUSS FOID Addendum.)

Virtual Component Linking Diagram:

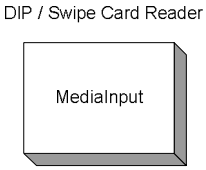


Figure 12: Linking for DIP / Swipe Card Reader

Description of Virtual Component Linkage:

A single MediaInput virtual component, configured as either a Swipe or DIP type of reader.

Distinct Characteristics:

MediaInput:	
Characteristic	**Value**
MediaType.MediaTypeListDef	MediaType.MediaTypeDef.MagneticStripe
MediaInput.typeOfReader	MediaInput.ReaderType.DIP MediaInput.ReaderType.Swipe
MediaInput.supportedDataTypes	DataType.MSG
MediaInput.numberOfTracks	<int>, depends on the actual number of tracks the hardware is capable of reading. Typically, this number is 2 or 3 for ISO readers
Manufacturer.FirmwareVersion	<string> - indication of the supported data type, such as: DS_TYPES_FOID_ISO, DS_TYPES_PAYMENT_ISO, DS_TYPES_DISCRETIONARY_ISO. See Chapter 8 for how to use these data types.

Supported extended data types:

By default, the CUSS card reader interfaces will only provide truncated track data to the application when the customer inserts a payment card. If an application requires full and legitimate access to the payment card information, if must call setup() to configure that access.

Please see the next section 7.8 and Chapter 8 for more information on extended data types and payment card data truncation, as well as a sequence diagram for this usage.

Typical Sequence Diagrams for DIP / SWIPE Magnetic Card Reader:

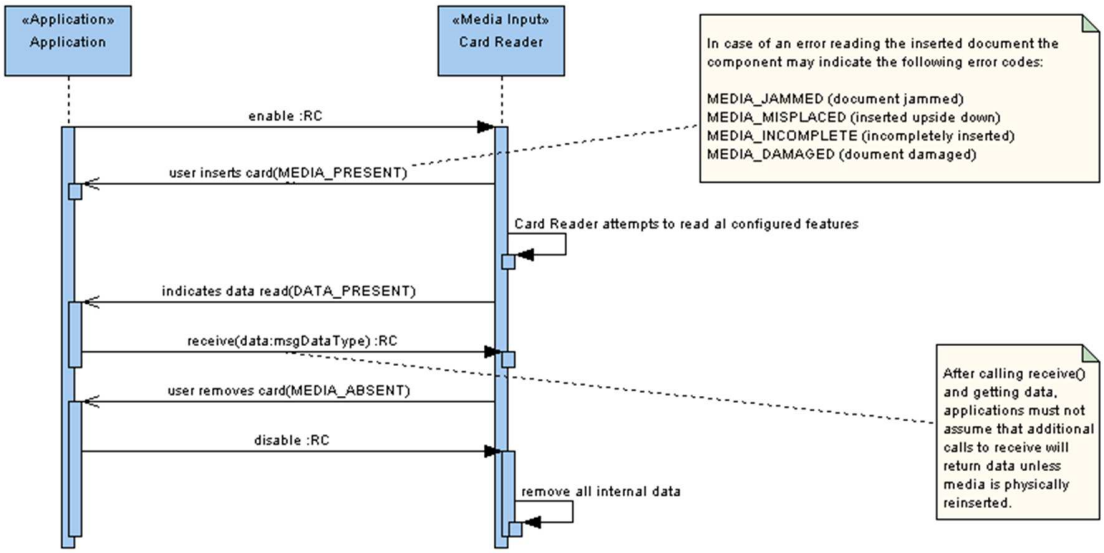


Figure 13: Reading Magnetic Cards on a DIP reader

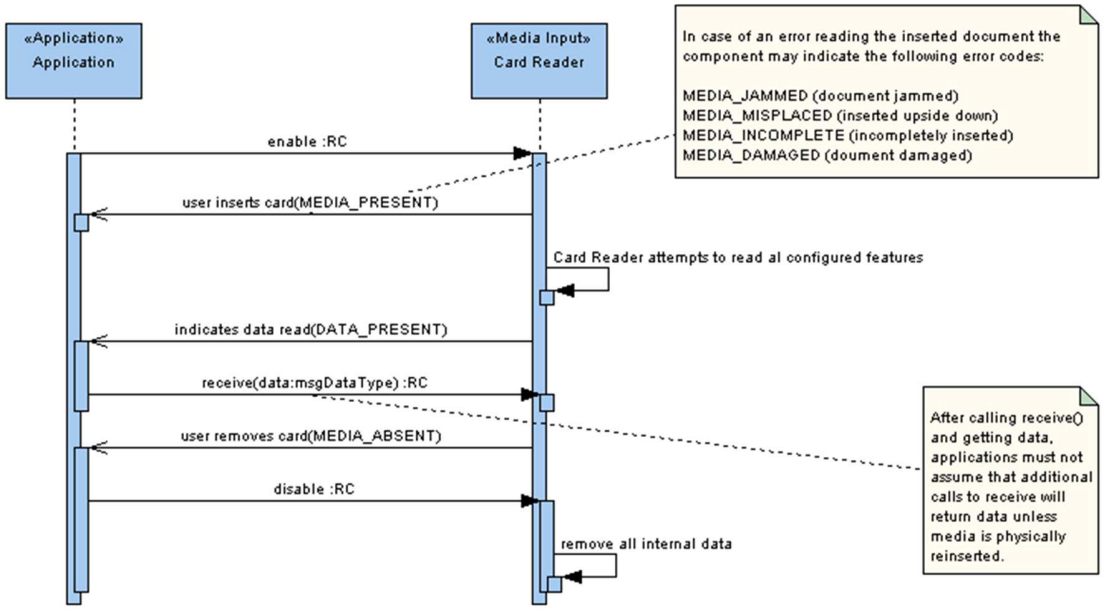


Figure 14: Reading Magnetic Cards on a SWIPE reader

7.9 Magnetic Card Encoder

Description of Device:

The motorized magnetic card encoder and reads and encodes magnetic cards that are inserted into the reader. There is a MediaInput and MediaOutput component linked to a real dispenser and a capture component. Depending on the actual hardware, the card encoder may support a number of different encoding formats / specifications. Each data type that can be read is represented by a MediaInput with the corresponding extended data type, and each data type that can be written is represented with a MediaOutput with the corresponding extended data type.

Virtual Component Linking Diagram:

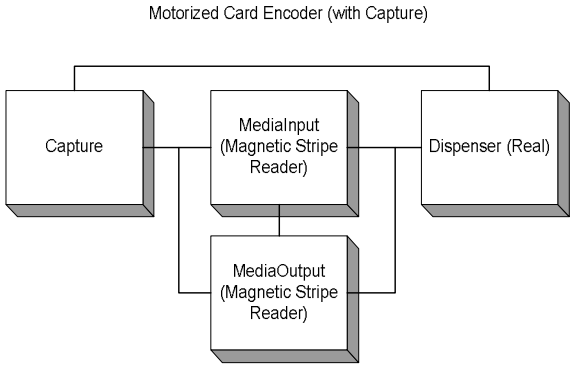


Figure 15: Linking for Motorized Magnetic Card Encoder (with Capture)

Description of Virtual Component Linkage:

The MediaInput and MediaOutput virtual components are linked to a real dispenser as the end user can only retrieve an inserted card after the offer() directive is called. The MediaInput and MediaOutput virtual components can also be linked to a capture virtual component, used to retain inserted cards. For devices that support capturing cards that are in the dispenser, the Dispenser virtual component can be linked to the Capture virtual component.

Distinct Characteristics:

MediaInput	
Characteristic	**Value**
MediaType.MediaTypeListDef	MediaType.MediaTypeDef.MagneticStripe
MediaInput.typeOfReader	MediaInput.ReaderType.Motorized
MediaInput.supportedDataTypes	DataType.MSG
MediaInput.numberOfTracks	<int>, depends on the actual number of tracks the hardware is capable of reading. Typically, this number is 2 or 3 for ISO readers
Manufacturer.firmwareVersion	Identifies the extended data types supported, based on CUSS Addendum A.1.34

MediaOutput	
Characteristic	**Value**
MediaOutput.type	MediaOutput.MediaType.Card
MediaType.MediaTypeListDef	MediaType.MediaTypeDef.MagneticStripe
MediaOutput.typeOfReader	MediaInput.ReaderType.Motorized

MediaOutput.supportedDataTypes	DataType.MSG
MediaOutput.numberOfTracks	<int>, depends on the actual number of tracks the hardware is capable of writing. Typically, this number is 2 or 3 for ISO writers
Manufacturer.firmwareVersion	Identifies the extended data types supported, based on CUSS Addendum A.1.34

Dispenser:	
Characteristic	**Value**
Dispenser.kind	Dispenser.DispenserType.real_ The dispenser is real because an offer() is required for the end user to retrieve the card.

Capture	
Characteristic	**Value**
Bin.BinSize	Maximum number of documents a bin can hold
Bin.AlmostFullLevel	Shows the high threshold of the bin if corresponding sensor is installed. May generate a MEDIA_HIGH event.
Bin.currentNoOfDocuments	Shows the current number of documents in the bin

7.10 Magnetic Card Encoder with Dispenser

Description of Device:

The motorized magnetic card encoder with dispenser reads and encodes magnetic cards that are inserted into the reader. There is a MediaInput and MediaOutput component linked to a real dispenser and a capture component. Depending on the actual hardware, the card encoder may support a number of different encoding formats / specifications. Each data type that can be read is represented by a MediaInput with the corresponding extended data type, and each data type that can be written is represented with a MediaOutput with the corresponding extended data type. The dispensers hold cards that can be encoded and offered to the user, similar to the way ATB printers can print coupons from its bins. For devices that support capturing cards that are in the dispenser, the Dispenser virtual component can be linked to the Capture virtual component.

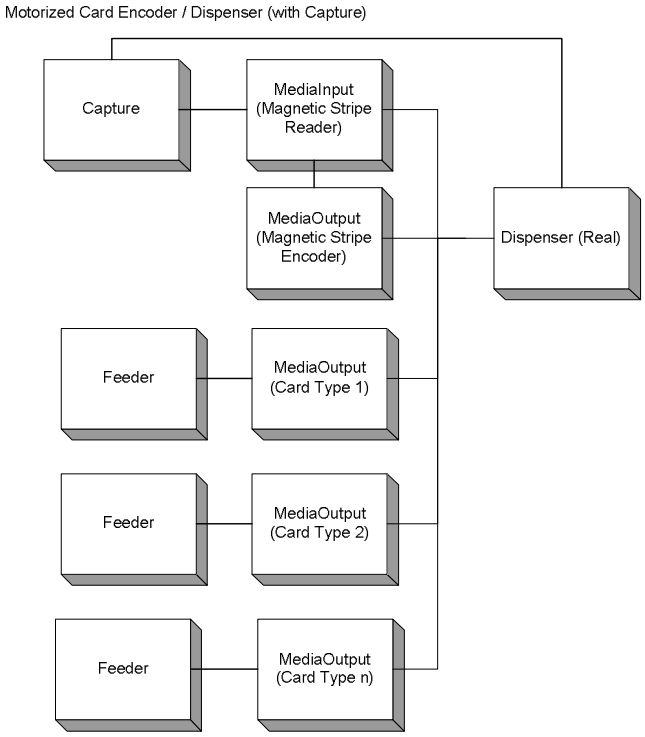


Figure 16: Linking for Motorized Magnetic Card Encoder (with Dispenser)

Description of Virtual Component Linkage:

The MediaInput and MediaOutput virtual components are linked to a real dispenser as the end user can only retrieve an inserted card after the offer() directive is called. The MediaInput and MediaOutput virtual components can also be linked to a capture virtual component, used to retain inserted cards. There is a MediaOutput component defined for each Feeder.

Distinct Characteristics:

MediaInput (Magnetic Stripe Reader)	
Characteristic	**Value**
MediaType.MediaTypeListDef	MediaType.MediaTypeDef.MagneticStripe
MediaInput.typeOfReader	MediaInput.ReaderType.Motorized
MediaInput.supportedDataTypes	DataType.MSG
MediaInput.numberOfTracks	<int>, depends on the actual number of tracks the hardware is capable of reading. Typically, this number is 2 or 3 for ISO readers
Manufacturer.firmwareVersion	Identifies the extended data types supported, based on CUSS Addendum A.1.34

MediaOutput (Magnetic Stripe Reader and Card Type 1.. n)	
Characteristic	**Value**
MediaOutput.type	MediaOutput.MediaType.Card
MediaType.MediaTypeListDef	MediaType.MediaTypeDef.MagneticStripe
MediaOutput.typeOfReader	MediaInput.ReaderType.Motorized
MediaOutput.supportedDataTypes	DataType.MSG
MediaOutput.numberOfTracks	<int>, depends on the actual number of tracks the hardware is capable of writing. Typically, this number is 2 or 3 for ISO readers
Manufacturer.firmwareVersion	Identifies the extended data types supported, based on CUSS Addendum A.1.34

Dispenser	
Characteristic	**Value**
Dispenser.kind	Dispenser.DispenserType.real_ The dispenser is real because an offer() is required for the end user to retrieve the card.

Capture	
Characteristic	**Value**
Bin.BinSize	Maximum number of documents a bin can hold
Bin.AlmostFullLevel	Shows the high threshold of the bin if corresponding sensor is installed. May generate a MEDIA_HIGH event.
Bin.currentNoOfDocuments	Shows the current number of documents in the bin

7.11 General Purpose Printer (GPP)

Description of Device:

The Simple General Purpose Printer (GPP) is a device that can print media based on the SVG format. It does not have an escrow device and once the coupons are printed, they are presented to the end user automatically. A platform provider can also choose to use the same set of virtual components to print both AEA and SVG data, represented by the supported data types in the virtual component characteristics. Another platform provider may choose to use a completely different set of virtual components to represent SVG and AEA data even if both sets of virtual components represent the same physical printer.

> **Important Notice:** depending on its capabilities, a CUSS kiosk may not include a GPP printer, it may include a single GPP interface, or may include two or more GPP printers for additional specialty functions such as Receipt printing or Heavy Tag printing (for Self Bag Drop devices.)

Virtual Component Linking Diagram:

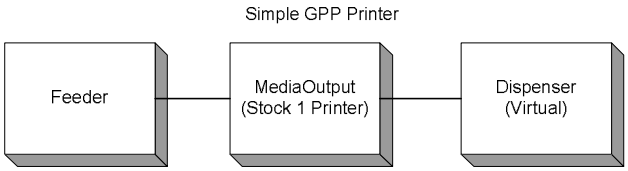


Figure 17: Linking for GPP

Description of Virtual Component Linkage:
A MediaOutput component is linked to a Feeder and a virtual Dispenser

Distinct Characteristics:

MediaOutput (Stock 1 Printer)	
Characteristic	**Value**
MediaType.MediaTypeListDef	MediaType.MediaTypeDef.Printed
MediaOutput.MediaType.type	MediaOutput.MediaType.GeneralPurposeDoc
MediaOutput.supportedDataTypes	DataType.SVG DataType.AEA (if the virtual components supports both AEA and SVG data)

Dispenser (Stock 1 Printer)	
Characteristic	**Value**
Dispenser.kind	Dispenser.DispenserType.virtual_ The dispenser is virtual because an offer() is not required. This is because once printed, the media is available to the end user. If offer() is called the platform will generate the response with status code OK so that application knows that the platform does not have a physical sensor in the dispenser component. If offer() is called against the virtual dispenser, the offer() must block until the tickets are removed if the platform supports the ability to detect removal of coupons.

MediaOutput.maxPrintSizeX MediaOutput.maxPrintSizeY	The width and height of the document in millimeters, used to distinguish multiple printers supporting different size paper.
MediaOutput.PrintOrientation	Indicates with the document is portrait (X narrower than height Y) or landscape (X wider than height Y)
MediaOutput.Manufacturer.firmwareVersion	May contain additional indications about the specialty nature of the documents printed by this GPP (for example, Heavy Tag adhesize stock for a self bag drop device)

Typical Sequence Diagram for GPP:

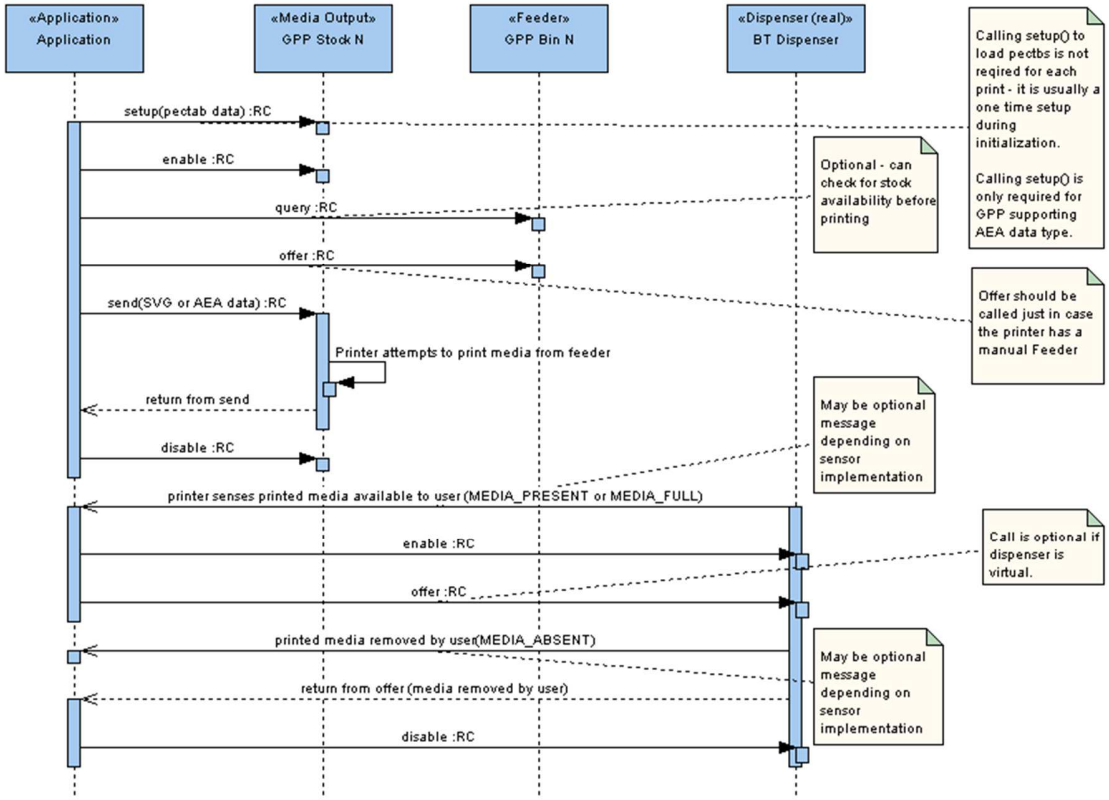


Figure 18: Reading Magnetic Cards on a SWIPE reader

7.12 DIP / Swipe Passport Reader

Description of Device:

The DIP / Swipe passport reader accepts and reads passports with OCR data. There is a single MediaInput component representing the DIP / Swipe Reader.

Virtual Component Linking Diagram:

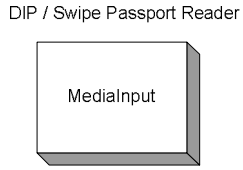


Figure 19: Linking for DIP / Swipe / Flatbed Passport Reader

Description of Virtual Component Linkage:

A single MediaInput virtual component, configured as either a Swipe / DIP type of reader.

Distinct Characteristics:

MediaInput	
Characteristic	**Value**
MediaType.MediaTypeListDef	MediaType.MediaTypeDef.Printed
MediaInput.typeOfReader	MediaInput.ReaderType.DIP MediaInput.ReaderType.Swipe
MediaInput.supportedDataTypes	DataType.MSG
MediaInput.numberOfTracks	<int>, depends on the actual number of OCR tracks
ComponentFonts.usedStandard	ComponentFonts.BarcodeStandard.nonApplicableBarcodeTypes

Typical Sequence Diagram for Passport Reader:

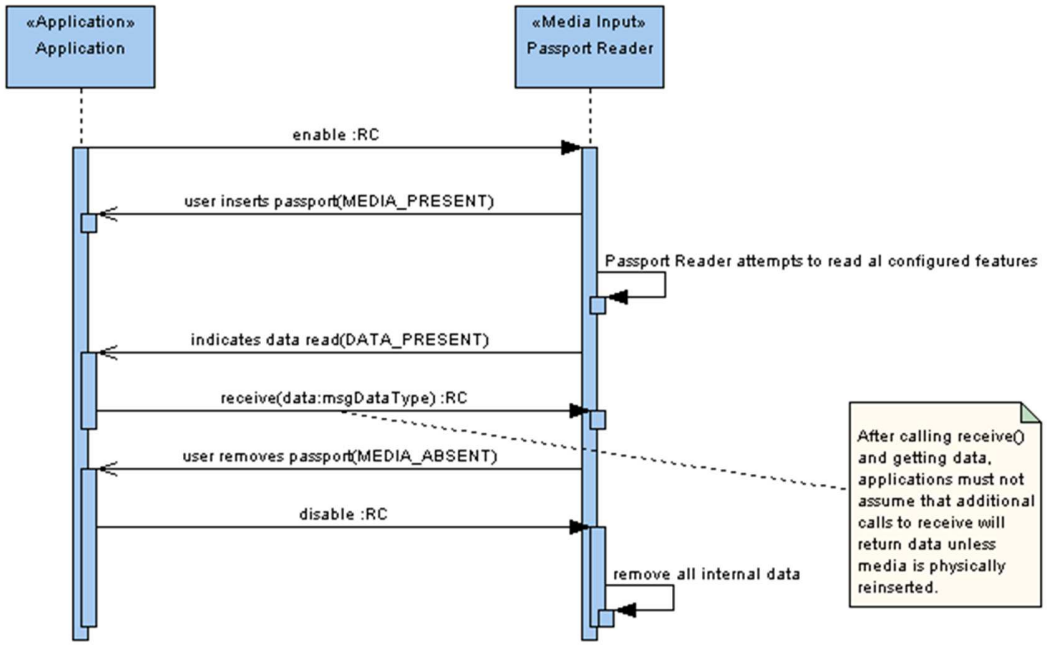


Figure 20: Reading Passports on a DIP or SWIPE reader

7.13 Barcode Scanner

Description of Device:

The barcode reader reads barcodes encoded using supported barcode technologies. There is a single MediaInput component representing the barcode scanner. OCR data is recorded in the data records within the MSG Data Type. If the barcode scanner supports multiple barcodes, the individual OCR data is stored in different data records within the MSG Data Type.

Virtual Component Linking Diagram:

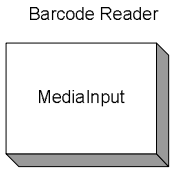


Figure 21: Linking for Barcode Reader

Description of Virtual Component Linkage:

A single MediaInput virtual component, configured as a one of the valid reader types

Distinct Characteristics:

MediaInput	
Characteristic	**Value**
MediaType.MediaTypeListDef	MediaType.MediaTypeDef.Printed
MediaInput.typeOfReader	MediaInput.ReaderType.PenScan MediaInput.ReaderType.Contactless MediaInput.ReaderType.Swipe MediaInput.ReaderType.FlatbedScan
MediaInput.supportedDataTypes	DataType.MSG
ComponentFonts.usedStandard	ComponentFonts.BarcodeStandard.Code39 ComponentFonts.BarcodeStandard.Code128 ComponentFonts.BarcodeStandard.Code2of5 The BarcodeStandard must be one of the above in order to differentiate it from a passport reader.
Manufacturer.FirmwareVersion	<string> should contain a list of supported barcodes, such as PDF417, etc.

Typical Sequence Diagram for Barcode Reader:

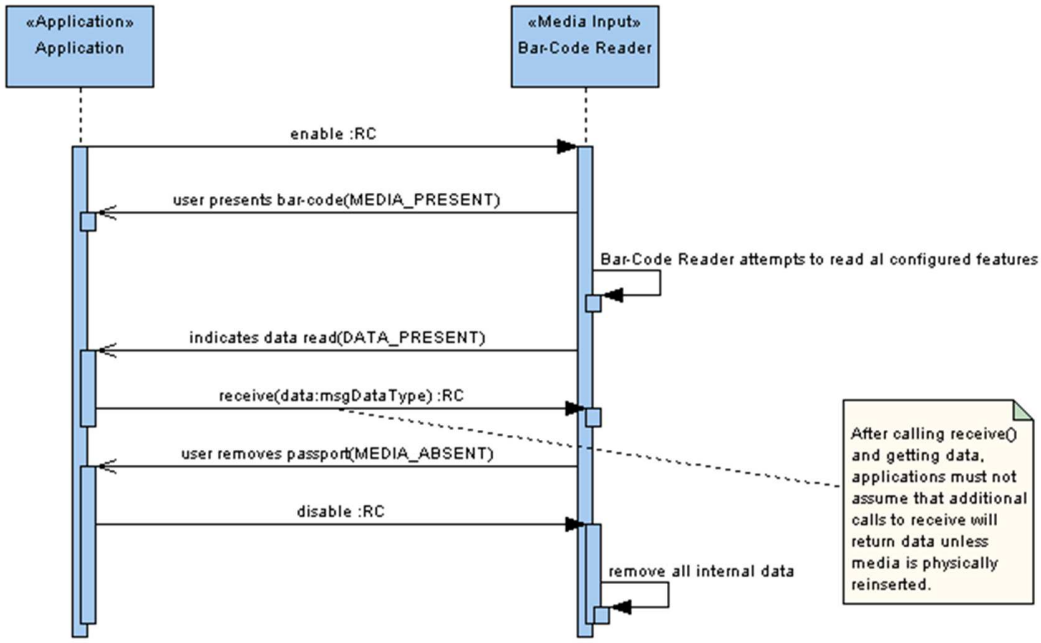


Figure 22: Reading bar-codes

7.14 Flatbed Reader

Description of Device:

The flatbed reader accepts and reads one or more of the following formats: OCR documents, barcode data, and image data. Due to the realization that a single device can read so many different types of data, each type of data that is supported by the device has its own MediaInput component associated with it. To differentiate the different MediaInput components, the firmware version field will contain the data type being supported. See the section 'Identification of extended media types supported by component' in the CUSS Addendum document for more details.

Virtual Component Linking Diagram:

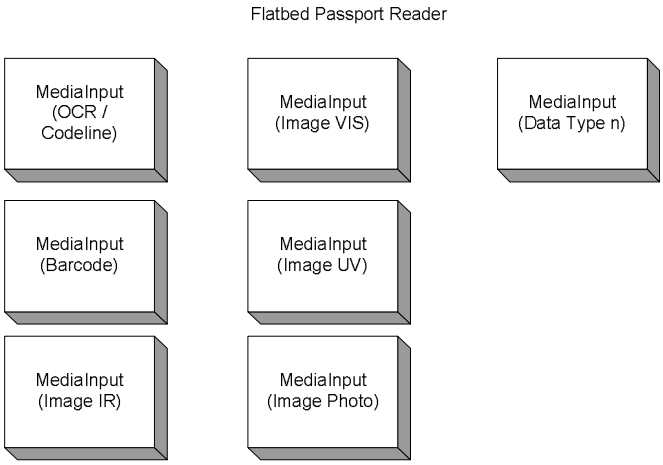


Figure 23: Linking for Flatbed Reader

Description of Virtual Component Linkage:

There is a single MediaInput virtual component for each configured valid data type that can be read by the reader. The components are not linked and are distinct, and from the application's perspective can be seen as individual devices. For example, if the flatbed reader is capable of reading both OCR data and barcode, it will appear to the application as a passport reader and a barcode reader. The characteristics will be identical, except for the firmware version indicating the data type and possibly the barcode standard.

Distinct Characteristics:

MediaInput (for all data types)	
Characteristic	**Value**
MediaType.MediaTypeListDef	MediaType.MediaTypeDef.Printed
MediaInput.typeOfReader	MediaInput.ReaderType.FlatbedScan
MediaInput.supportedDataTypes	DataType.MSG
ComponentFonts.usedStandard	ComponentFonts.BarcodeStandard.nonApplicableBarcodeType ComponentFonts.BarcodeStandard.Code39 ComponentFonts.BarcodeStandard.Code128 ComponentFonts.BarcodeStandard.Code2of5
Manufacturer.FirmwareVersion	<string> - indication of the supported data type, such as: DS_TYPES_CODELINE, DS_TYPES_BARCODE,

	DS_TYPES_IMAGE_PHOTO, DS_TYPES_IMAGE_COAX, DS_TYPES_IMAGE_UV, DS_TYPES_IMAGE_VIS, DS_TYPES_IMAGE_IR, etc. See the section 'Identification of extended media types supported by component' in the CUSS Addendum document for more details

7.15 RFID/NFC/Contactless Media Reader

Description of Device:

The RadioRFID reader accepts and reads Proximity Integrated Circuit Cards (PICCs) which conforms to one of the following transport standards:

- ISO 10536
- ISO 14443
- ISO 15693
- ISO 18092

These PICCs communicate with these protocol standards:

- ISO 7816
- MIFARE

Due to the realization that a single device can read so different types of data, but only one at a time, the device is implemented as on single MediaInput component supporting all types of data associated with it. Review Chapter 6 and Appendix H for more information on identifying device components which support specific types of data. Addition transport and protocol standards may be supported by other readers and cards. The same concepts in this Section would apply to them in general terms as well.

Here is a description of the behaviour of these types of devices:

1. After card detection the virtual component1 platform will inform the active application with MEDIA_PRESENT event. The MediaInput component will select the card using the appropriate command(s) after card detection.

2. Answer To reset (ATS) will be signaled with DATA_PRESENT and can be read using the receive directive.

3. In case multiple cards have been detected by the platform, the platform will send a MEDIA_MISPLACED event. It's up to the application to ask the user only to tap a single card at a time. Anti-collision functionality has only to be supported in the sense of detecting the presence of multiple cards.

4. Commands will be sent to the chip with the setup directive using an msgDataType. For performance reasons, commands can be bundled. Each record of the msgDataType represents a single command. The platform is responsible to process them sequentially, in the order they appear in the message starting with records[0].message, records[1].message, etc.

5. Regardless of the communication mode (synchronous or asynchronous) the setup directive will not return any data in the event data. If the setup directive is called in blocking mode, it will return after either all commands have been executed or the timeout has expired.

6. Applications have to consider that the specified timeout for the setup directive defines the overall timeout for all commands in the bundle.

7. The presence of the result data from a previous setup directive call will be signaled with a DATA_PRESENT event. The data can be read using the receive directive.

8. Then calling the setup directive asynchronously and the receive directive synchronously, it's the applications responsibility to select appropriate timeout values. If not the platform behavior is undefined.

Supported extended data type DS_TYPES_ISO7816:

C-APDUs (**C**ommand **A**pplication **P**rotocol **D**ata **U**nit) can be bundled. Each record of the msgDataType represents a single C-APDU. R-APDU (**R**esponse **A**pplication **P**rotocol **D**ata **U**nit) will be sent to the application under the same record number as the corresponding C-APDU. A dataRecord of the msgDataType has the following structure:

```
// C-APDU
record[0].message[0..n] = C-APDU

// R-APDU
record[0].message[0..m] = R-APDU
// SW1 (Status Word 1)
record[0].message[m+1]  = SW1
// SW2 (Status Word 2)
record[0].message[m+2]  = SW2
```

Supported extended data type DS_TYPES_MIFARE:

The principle is the same as for ISO7816 compliant cards. PICC commands a data block or value block can be mixed and bundled. Each dataRecord of the msgDataType represents a single PICC command. A record of the msgDataType has the following structure:

```
// PICC Command:
//   Authenticate using KeyA = 0x60
//   Authenticate using KeyB = 0x61
//   Read Data Block         = 0x30
//   Write Data Block        = 0xA0
//   Decrement Value Block   = 0xC0
//   Increment Value Block   = 0xC1
//   Restore Value Block     = 0xC2
record[0].message [0] = PICC command

// Block number to use
record[0].message [1]        = block number

// Key (48 bits) to use for Authenticate commands (0x60 & 0x61)
record[0].message [2..7]     = Key to use

// Value to use for Value Block commands (0xC0, 0xC1 & 0xC2)
record[0].message [2..5]     = Value
record[0].message [6]        = Transfer Address
```

```
// Value to use for Data Block commands (0xA0)
record[0].message [2..17]    = data to use
```

Other commands will be implicitly handled by the platform. E.g. each value block operation will be followed by a Transfer Value Block (0xB0) operation. Anti collision commands cannot be sent by the kiosk application.

Performance considerations:

The Dwell period of the platform must not be greater than 50ms. The dwell period is defined as the time receiving the setup message until the response from the PICC has been processed and the DATA_PRESENT event has been send to the calling CUSS application minus the processing time of the PICC.

Virtual Component Linking Diagram:

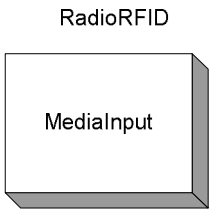


Figure 24: Linking for RadioRFID Reader

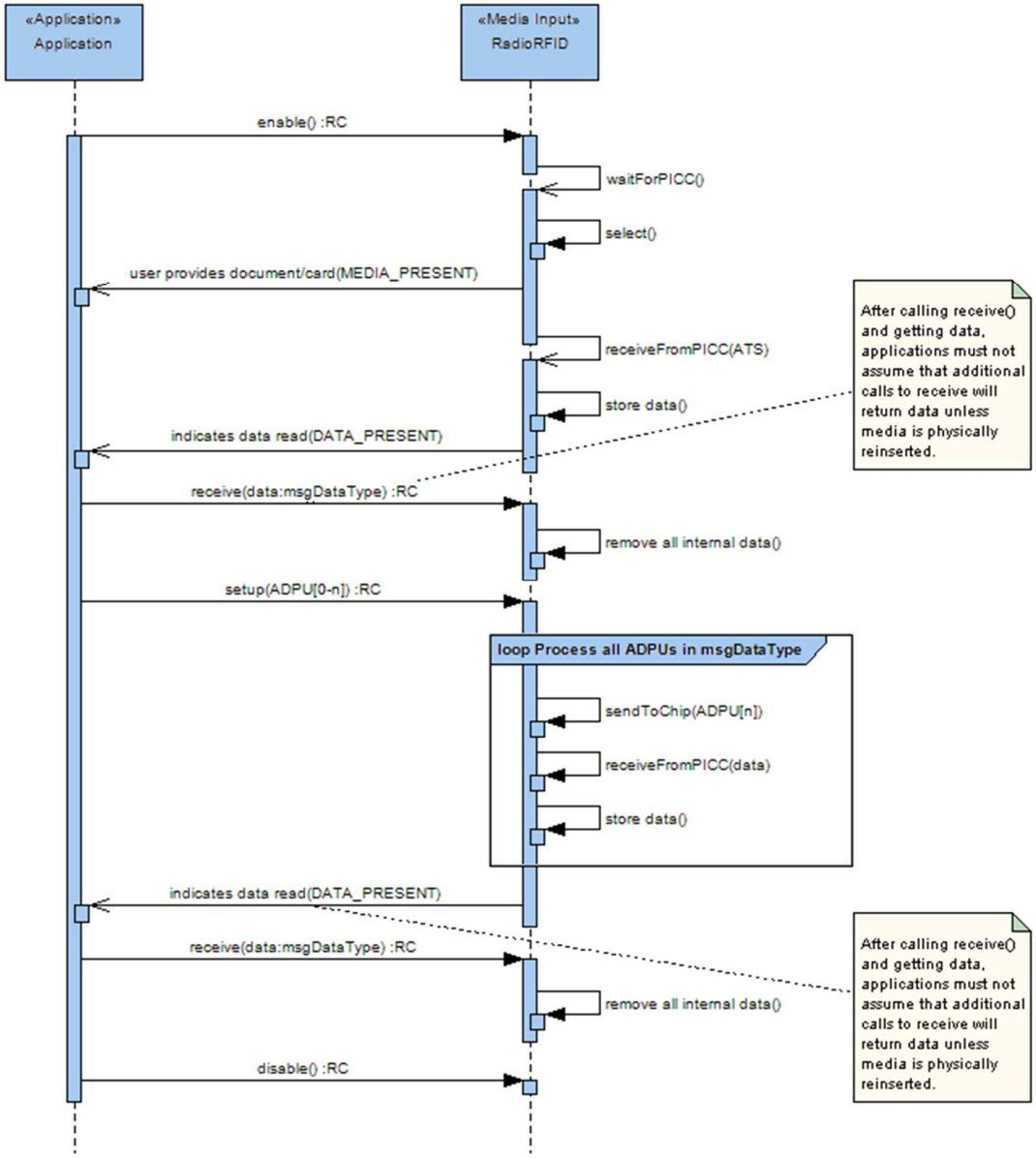
Description of Virtual Component Linkage:

There is a single MediaInput virtual component for all configured valid data type that can be read by the reader.

Distinct Characteristics:

MediaInput (for all data types)	
Characteristic	**Value**
MediaType.MediaTypeListDef	MediaType.MediaTypeDef.Chip
MediaInput.typeOfReader	MediaInput.ReaderType.Contactless
MediaInput.supportedDataTypes	DataType.MSG
ComponentFonts.usedStandard	
Manufacturer.FirmwareVersion	<string> - indication of the supported data type, such as: DS_TYPES_ISO7816, DS_TYPES_MIFARE and <string> indication of the supported transport standards, such as: ISO10536, ISO14443, ISO15693, ISO18092 etc.

Typical Sequence Diagram for RadioRFID Reader:


«Application»
Application
«Media Input»
RadioRFID
enable() :RC
waitForPICC()
select()
user provides document/card(MEDIA_PRESENT)
receiveFromPICC(ATS)
store data()
indicates data read(DATA_PRESENT)
After calling receive() and getting data, applications must not assume that additional calls to receive will return data unless media is physically reinserted.
receive(data:msgDataType) :RC
remove all internal data()
setup(ADPU[0-n]) :RC
loop Process all ADPUs in msgDataType
sendToChip(ADPU[n])
receiveFromPICC(data)
store data()
indicates data read(DATA_PRESENT)
receive(data:msgDataType) :RC
remove all internal data()
After calling receive() and getting data, applications must not assume that additional calls to receive will return data unless media is physically reinserted.
disable() :RC

7.16 Integrated Baggage System (Self Bag Drop AEA-SBD)

Important Note:

In CUSS 1.3 there are two defined methods of using Self Bag Drop conveyor devices.

- AEA-SBD (Section 7.16) allows complete control using the AEA2012-2 specification for bag drop devices
- CUSS-SBD (Section 7.17) allows complete control using the CUSS traditional virtual component model

CUSS platforms must implement both interface options if running on a self-service kiosk that includes a Self Bag Drop conveyor.

CUSS applications must only use one of the interface options when attempting to control Self Bag Drop conveyor on the kiosk. An attempt to initialize both interfaces will result in an error: the platform shall return RC_DENIED if the application calls acquire() and another interface has already been acquired.

Description of Device:

Integrated Baggage Systems are complex devices allowing passengers to check-in their baggage themselves. Typically these devices are made from a set of separate devices for weighing and checking dimensions of the introduced bags as well as validating printed and attached baggage tags before these bags are fed into the airports baggage sortation systems. Scanning and validating baggage tags are typically based on the license plate definitions defined in the IATA Resolution 740 and may be also supported by RFID antennas, but the specification does not restrict barcode formats used.

Integrated Baggage Conveyors systems are not intended to X-ray baggage for explosives or other security critical items as this task is usually done prior or after the baggage check-in process.

An Integrated Baggage System always includes conveyers and integration into a baggage sortation system. For dedicate weight scales for baggage that are not conveyors, see Section 7.17. A kiosk may be connected to a Baggage Scale and an Integrated Baggage Conveyor at the same time.

Important Note: **The Integrated Baggage System interface does not replace the CUSS interfaces for card readers, passport readers, and document scanners. CUSS platforms must provide the normal CUSS component interfaces for the devices, if equipped at the position.**

In addition to the CUSS interfaces, the platform may also provide optional access using the AEA-SBD interface, but this is not required.

Important Note: This component definition ***extends*** the existing Conveyor component definition in Section 7.17 below. Kiosks that include Self Bag Drop devices with a CUSS 1.3 platform **must provide both this AEA-SBD interface component as well as the complete CUSS Conveyor component grouping.**

Information and Background:

As indicated above, a previous **Conveyor** interface definition existed in CUSS-TS 1.2. Since 2009, new types of Self bag Drop devices and capbalities highlighted gaps in this Conveyor interface, meaning a change was needed for CUSS-TS 1.3.

While the previous interface was designed around the CUSS "Standard Mode" virtual component model, the new interface uses a direct command and control interface based on AEA-SBD, instead of a detailed multi-component model.

The reason for this change is the requirement from the Common Use community (airlines) that individual standards such as CUSS and CUPPS not define their own native interfaces to integrated bag drop devices, but use a common approach as encapsulated in the AEA2012-2 SBD specification.[43]

This CUSS-TS 1.3 maintains the existing component-based Conveyor model but adds and a simple UserOutput component model that uses AEA-SBD as the control mechanism, in accordance with the industry wish for commonality. The anticipated benefits of this approach are:

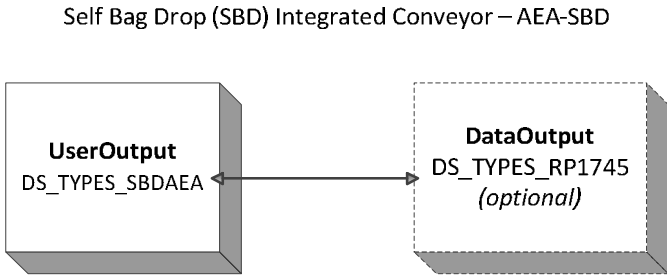
- A single effort to define a consistent specification within AEA-SBD, instead of individual effors within CUSS, CUPPS, and AEA.
- Shortcomings aqnd ambiguities are resolved across the airline and airport community instead of being isolated in specific working groups such as TSG-CUSS.
- Maintains the existing Conveyor interface from CUSS 1.2 (modified to meet more current needs) to ensure that those with existing CUSS SBD investments do not require a complete update to their applications.
- Common use application development efforts will lower given that Self Bag Drop logic will be similar across common use kiosk and agent applications, as well as proprietary applications.
- The AEA-SBD specification is easer to adapt and modify in response to industry needs as opposed to isolated CUSS or CUPPS specifications.

The AEA-SBD specification is a separate standard published by and available by subscription from the Association of European Airlines. It is not included with or part of the IATA CUSS Technical Specification.

http://www.aea.be/research/specs/index.html

[43] For details, review meeting minutes from CUWG meetings in Orlando, USA, May 2012.

Virtual Component Linking Diagram:



A CUSS application that chooses to use the AEA-SBD interface shall only acquire() the UserOutput and (if present and needed) the DataOutput components listed above, and shall not acquire any of the other components related to the CUSS-SBD interface (Conveyors, etc.) The platform will respond RC_DENIED if an attempt is made to acquire a CUSS-SBD component after an AEA-SBD component has already been acquired, and vice versa.

Distinct Characteristics:

UserOutput	
Characteristic	**Value**
Manufacturer.FirmwareVersion	This will include the indicator DS_TYPES_SBDAEA to confirm the device represents an Integrated Baggage Conveyor supporting the AEA2012-2 SBD extensions.

DataOutput (airport BHS) -- *optional*	
Characteristic	**Value**
Manufacturer.FirmwareVersion	This will include the indicator DS_TYPES_RP1745 to indicate that RP1745-compliant BSMs are supported by this data interface.

Distinct Status Conditions:

The platform shall return RC_DENIED when an application attemps to call acquire() on an AEA -SBD component after the application has already acquired a CUSS-SBD component

7.16.1 Data Format (DS_TYPES_SBDAEA)

All requests for the Integrated Baggage Conveyor UserOutput component using setup() and send() requests must be constructed in accordance with the *Self Baggage Drop (SBD)* definition in AEA2012-2 (or, a later version of AEA if specifically indicated in that later version of AEA.)

For more information on the command and response protocol for SBD devices in AEA2012-2, refer to the official AEA documentation available from the Association of European Airlines (AEA.) Reference information is included below.

The AEA specification for Self Bag Drop is published by the Association of European Airlines (AEA) and is obtained by subscription purchase. It is not maintained, published, or available from IATA. Please review the following website for more information.

http://www.aea.be/research/specs/index.html

The following quick reference table is supported by the CUSS Integrated Baggage Conveyor component:

CUSS command restrictions		*As supported by the equipment*		
CUSS	UserOutput::**setup()**	**PV, AV, CT, EP, ES, RI, RC, IS, IX, IL, VS, VX, VL, LS, LC, LT**		
CUSS	UserOutput::**send()**	**SQ, BQ, CW, CR, CC, CB, CE, MG, LA, RD**		
Command	**Purpose**	**Description**	**Category**	**Section**
Device Capabilities and Information		*Applies to all SBD devices*		
PV	Program Version	Returns the program/firmware version in place on the SBD	Detection/Configuration	2.49
AV	AEA Version	Returns the version of the AEA specification supported by the SVD	Detection/Configuration	2.38
CT	Code Transaction	Sets the 1-5 letter transaction code for AEA messages	Detection/Configuration	2.47
EP	Environment Program	Tells the SBD what operating parameters to use (weight units, etc.)	Detection/Configuration	2.1
SBD Operating Parameters		*Applies to all SBD devices*		
ES	Environment Status	Returns the SBD's current operating parameters	Detection/Configuration	2.2
RI	Read Information	Returns detailed information about the operation of the SBD	Detection/Configuration	2.34
RC	Read Configuration	Returns the SBD's current equipment level and configuration	Detection/Configuration	2.4
Status and Bag Monitoring		*Applies to all SBD devices*		
SQ	Status Query	Provides status condition of the individual components within the SBD	Status Monitoring	2.6
BQ	Bag Query	Return all information about bags inside the SBD	Status Monitoring	2.13
SQNI	Status Query (unsoliticted)	Asynchronous notification version of SQ	Status Monitoring	2.8
BQNI	Bag Query (unsolicited)	Asynchronous notification version of BQ	Status Monitoring	2.22
Conveyor Control		*Applies to all SBD devices*		
CW	Conveyor Wait	Stops SBD operation and, if equipped, closes and locks the access door	Mechanical Control	2.9
CR	Conveyor Resume	Starts SBD operation and, if equipped, unlocks and opens access door	Mechanical Control	2.11
CC	Conveyor Control	Process a bag or move it between belts or to the airport BSS	Mechanical Control	2.23
CE	Cancel Bag (LED)	Cancels the bag process, returns bags, and indicates the LED	Mechanical Control	2.41
CB	Cancel Bag (LED/Buzzer)	Cancels the bag process, returns bags, and indicates the LED and buzzer	Mechanical Control	2.44
Passenger Display Screen Control		*Only where equipped*		
LA	Lighting and Audio	Controls LED/light and audio operations on the SBD	User Indicators	2.36
MG	Message Graphics	Provides text, graphics, or video to display on optional screen	User Indicators	3.1
IS	Image Status	Returns the list of images currently loaded in system	Status Monitoring	3.4
VS	Video Status	Returns the list of videos currently loaded in system	Status Monitoring	3.5
IX	Image Clear	Clears all images loaded in the system	Detection/Configuration	3.6
VX	Video Clear	Clears all videos loaded in the system	Detection/Configuration	3.7
IL	Image Load	Load an image into the system	Detection/Configuration	3.8
VL	Video Load	Load a video into the system	Detection/Configuration	3.9
Receipt Printer Control		*Only where equipped*		
RD	Receipt Document	Sets the receipt data to print on the receipt printer in the SBD	Mechanical Control	4.5
LT	Logo Type	Sets logo data to use on receipt printer	Detection/Configuration	5.1
LS	Logo Status	Returns a list of all receipt printer logos loaded in the system	Status Monitoring	5.4
LC	Logo Cancel	Clears logo(s) loaded in the system	Detection/Configuration	5.7
Extended Data Notification		*Only where equipped*		
BCRI	Bar Code Reader Info	Provides barcode scanner data for non-bagtag reader in the SBD	Status Monitoring	6.1
OCRI	Optical Charater Reader Info	Provides OCR MRZ scanner data for reader integrated in the SBD	Status Monitoring	6.2
MSRI	Mag Stripe Reader Info	Provides multi-track magnetic card data for reader integrated in the SBD	Status Monitoring	6.3
RFDI	Radio Frequency Document Inf	Provides RFID data for e-Passport reader integrated in the SBD	Status Monitoring	6.4
RFRI	Radio Frequency Reader Info	Provides RFID data for radio reader integrated in the SBD	Status Monitoring	6.5
NFCI	Near Frequency Comm Info	Provides data for NFC reader integrated in the SBD	Status Monitoring	6.9
BIOI	Biometric Information	Provides data for Biometrics reader integrated in the SBD	Status Monitoring	6.1
CAMI	Camera Information	Providers data for snapshot/picture reader integrated in the SBD	Status Monitoring	6.11

The CUSS standard limits which AEA commands an application is permitted to send to the Integrated Baggage Conveyor, in order to maintain the state of the device for all applications.

Please refer to the AEA2012-2 SBD specification directly for information on the differences between soliticed and unsolicited events, for example SQOK responses compared to SQNI responses to the SQ command.

Unsolicited AEA messages such as SQNI, must be reported to CUSS applications as DATA_PRESENT events sent to the event listener registered by the application for the UserOutput components. As this component is not an Input component and does not support the receive() directive, the AEA message information shall be included in the event datastream field in the aeaDataType format.

To ensure that CUSS can use Self Bag Drop devices and future versions of AEA-SBD, **there are no restrictions on which commands an application may issue to SBD devices.** As of AEA2012-2, the following commands are defined with the AEA specification and will be supported by the platform:

Self bag Drop (SBD) devices (AEA2012-2 or later):

setup() directive:

PV, AV, CT, EP, ES, RI, RC, IS, IX, IL, VS, VX, VL, LS, LC, LT

send() directive:

SQ, BQ, CW, CR, CC, CB, CE, MG, LA, RD

Important Notice: certain AEA-SBD commands or command parameters, for example environment settings requests via the ES command, may need to be restricted by the platform in order to protect the integratity and operation of the SBD device in a shared environment.

In those cases where the platform needs to restrict the request, it shall return the appropriate AEA-SBD response string indicated for the failure of the requested command, if defined, or ERR7 if not defined.

7.16.2 Data Format (DS_TYPES_RP1745)

The Baggage Handling System (BHS) component is an *optional* component of the CUSS-SBD interface.

BHS DataOutput components that support communication via Baggage Source Messages (BSM) will report this capability by including the DS_TYPES_RP1745 Characteristic.

All BSM requests for the Baggage Handling System DataOutput component using send() must be standard Baggage Source Messaging as defined in IATA Recommended Practice RP1745 and must contain at least elements .V, .F and .N.

For more information on the specification for BSMs, please review IATA Recommended Practice 1745-Baggage Information Messages. Here is an example message:

```
BSM
```

```
.V1LFRA
.F/LH123/15MAR/BCN/F
.N0220567890001
.PGEHLING/AMR
ENDBSM
```

Not all BHS components will support BSMs. BHS components that do not support BSMs are typically used only for monitoring the condition of the airport baggage system/belt. There are no dedicated BHS status codes defined in the CUSS Technical Specification; the availability of the BHS shall be reported using existing CUSS virtual event and status codes.

7.16.3 Important Information and Clarifications

Where do I find the AEA-SBD Technical Specification?

The AEA specification for Self Bag Drop is published by the Association of European Airlines (AEA) and is available by subscription purchase. It is not maintained, published, or available from IATA. Please review the following website for more information.

http://www.aea.be/research/specs/index.html

How do I suggest changes or clarifications to the the AEA-SBD Technical Specification?

Please contact the AEA group listed above, which controls the AEA-SBD specification. The IATA CUSS Technical Solution Group may also make direct requests to AEA on behalf of the CUSS technical community, if appropriate to ensure the consistency of CUSS implementations of AEA-SBD.

Does the enable() directive immediately activate the AEA-SBD device?

No. A self bag drop is a collection of multiple components including conveyors, scales and belts. The CUSS enable() directive does not physically activate any specific component of the SBD. It only allows the application to send AEA-SBD commands to manipulate the individual components, such as the CR command to enable the conveyor and drop point.

Does the AEA-SBD interface support reading multiple License Plate barcodes at once?

Yes, the SBD interface allows the conveyor device to read and report more than one license plate barcode in its scanning area. This information is reported to the application as multiple BQ information sequences and similar. Refer to the AEA-SBD specification for more details.

Does the AEA-SBD interface support reading multiple other barcodes at once?

No, the SBD currently (as of AEA2012-2 SBD) supports reading only a single barcode using its boarding pass or handle held scanner.

How is RFID data encoded for the RFID reader capability of the AEA-SBD?

The current AEA2012-2 SBD specification does not list an encoding format; however, the data is encoded in base64 format. A future update to AEA-SBD will include more information on RFID formatting.

Applications that need to read RFID in a well-defined structure should use the alternate CUSS-SBD interface described in 7.17.

Does the AEA-SBD interface support RFID Writers/Encoders?

No, the current AEA2012-2 SBD specification only includes support for RFID readers. A future update to AEA-SBD may add RFID encoding features.

Applications that need to encode RFID in the current specification, should use the alternate CUSS-SBD interface described in 7.17.

How do I determine where the reader components are, such as the scale, license plate scanner, I order to properly read data from the bags.

There is no method in the current AEA2012-2 SBD to determine the specific configuration and design of the SBD device, such as where or how it reads bag weight, verifies bag dimensions, scans for attached tags, and similar.

This information is hidden from the application since a wide range of hardware designs exists, and it is unreasonably to impose that applications track and control the devices at that level.

This, the applications must use the “process bag” request (CP) and it is a platform and/or SBD device requirement to properly manipulation and position the bag (if needed) to read the data and return it the the application. It is not an application responsitily.

Why are the Light, Audio, Screen and Receipt components included in AEA-SBD and not as native CUSS interfaces?

The AEA-SBD specification is designed around fully-integrated self bag drop systems that include all the features needed to carry out a bag drop transaction. To allow this, the AEA-SBD specification lists certain *optional* components:

- Light/LED indicator
- Beeper/buzzer
- Screen capable of displaying text, graphics, and animations
- Receipt printer for baggage receipts

These component types do not currently have CUSS interface definitions. Since these sub components are designed to be an integral part of a bag drop machine and they are defined as optional in AEA-SBD, applications must control them using AEA.

In addition, the AEA-SBD command specification allows the application to “inline” requests for multiple sub components into a single command, for example combining a CC (conveyor control), RD (receipt

print) and LA (light/audio) requests into a single AEA command. For this reason, the components must remain integrated in CUSS.

The AEA-SBD specification includes support for MRZ, barcode, RFID, and other media readers. Should I use the AEA interface, or will these devices continue to be available though CUSS standard component interfaces?

The AEA-SBD specification is designed around fully-integrated self bag drop systems that include all the features needed to carry out a bag drop transaction. To allow this, the AEA-SBD specification lists capabilities for very advanced devices that include *optional* components:

- Passport/MRZ reader
- Barcode reader
- Biometric reader
- RFID/NFC reader
- Snapshot/camera
- Receipt printer

If these devices are available for use on a self-service baggage drop off point, for transactions other than bag drop operations (for example, check-in or doc check) then the CUSS platform must allow the application to control the device using the CUSS standard component mode as described extensively in this document. The CUSS platform may also choose to expose the components using AEA-SBD *in addition* to the CUSS standard implementation of these devices.

To be very clear:

- If a kiosk running a CUSS 1.3 platform includes a self-bag drop device which includes a card reader, and the kiosk does not include its own separate card reader, then **the CUSS platform must implement the Card Reader interface as defined in Section 7.7 or Section 7.8 to control the card reader integrated into the SBD.**

- If a kiosk running a CUSS 1.3 platform includes a self-bag drop device which includes a customer-facing barcode scanner, and the kiosk does not include its own separate barcode scanner, then **the CUSS platform must implement the Barcode Scanner interface as defined in Section 7.13 to control the customer-facing barcode scanner integrated into the SBD.**

- If a kiosk running a CUSS 1.3 platform includes a self-bag drop device which includes a MRZ document reader, and the kiosk does not include its own separate document reader, then **the CUSS platform must implement the document reader interface as defined in Section 7.12 or 7.14 to control the customer-facing barcode scanner integrated into the SBD.**

- If a kiosk running a CUSS 1.3 platform includes a self-bag drop device which includes a BTP bag tag printer, and the kiosk does not include its own separate bag tag printer, then **the CUSS platform must implement the bag tag printer interface as defined in Section 7.6 to control the customer-facing bag tag printer integrated into the SBD.**

- If a kiosk running a CUSS 1.3 platform includes a self-bag drop device which includes a receipt printer, and the kiosk does not include its own separate General Purpose Printer (GPP), then **the CUSS platform must implement the GPP interface as defined in Section 7.12 or 7.14 to control the receipt printer integrated into the SBD.**

- *This requirement does not apply if the integrated SBD receipt printer is not capable of general purpose printing.*

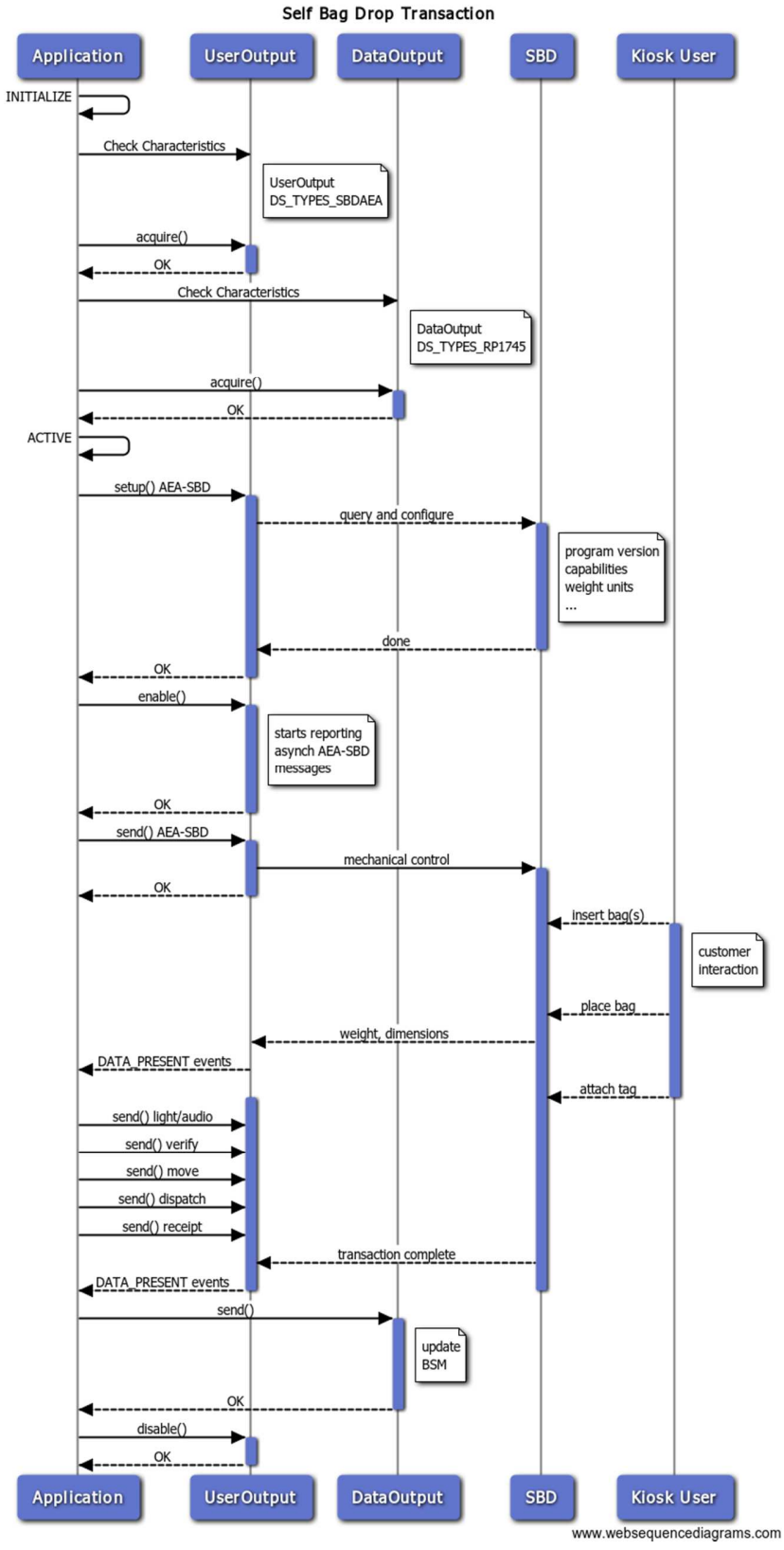
- In all cases, a CUSS platform may choose to provide access to these devices via the AEA-SBD protocol, in addition to the CUSS component mode access mandated above, so long as this is done in a way where both types of interface behave according to their respective specifications.

7.16.4 AEA-SBD Command and Control Examples

The AEA-SBD control protocol for Self Bag Drop devices is used in CUSS kiosks, CUPPS workstations, and in proprietary systems. For this reason knowledge and expertise regarding AEA-SBD command sequence extends beyond this CUSS Technical Specification.

For information on appropriate usage of the AEA-SBD specification, the CUSS Technical Specification thus does not attempt to create a reference document here and defers to the AEA specification and community itself, or other working areas within the Common Use community

7.16.5 Typical Sequence Diagram (AEA-SBD component)



7.16.6 Receipt and Heavy Tag Printing

Please refer to section 6.4 for information on how specialty document printing, such as baggage receipts and heavy tag printing, should be accomplished using the existing CUSS General Purpose Printer (GPP) capability for SVG and PDF documents.

These specialty GPP printers will be their own component group and not linked to the SBD components.

In particular, if a CUSS kiosk supports heavy tag printing or baggage receipt printing as part of its self bag drop device, the CUSS platform shall:

1. Include a GPP printer definition as set in Section 7.11 for each specialty printer
2. Ensure that the components' Characteristics about paper size and orientation are accurate for each printer
3. Include the characteristics keyword DS_TYPES_HEAVYTAG as part of a heavy tag printer component's Manufacturer.firmwareVersion setting.

As well, a kiosk provider shall publish information about the formatting and size requirements for their heavy tag printer to all airlines deploying on the SBD kiosk. This is important because, **at the time of publication of this CUSS Technical Specification there is no industry standard for specialty/heavy tag printing.**

For receipt printing, if the AEA-SBD implements a receipt printer interface on a device that is capable of general purpose printing (not a line printer), the kiosk platform must also implement a CUSS GPP interface for that receipt printer.

7.17 Integrated Baggage System Conveyor (CUSS-SBD)

Important Notices:

Though it is similar, **the CUSS-SBD interface is not backwards compatible with the previous Conveyor interface** in CUSS-TS 1.1 and 1.2. Applications and platforms will both need upgrades to operate as CUSS-TS 1.3 compliant components. See below under 7.17.3 for more details.

In CUSS 1.3 there are two defined methods of using Self Bag Drop conveyor devices.

- AEA-SBD (Section 7.16) allows complete control using the AEA2012-2 specification for bag drop devices
- CUSS-SBD (Section 7.17) allows complete control using the CUSS traditional virtual component model

CUSS platforms must implement both interface options if running on a self-service kiosk that includes a Self Bag Drop conveyor.

CUSS applications must only use one of the interface options when attempting to control Self Bag Drop conveyor on the kiosk. An attempt to initialize both interfaces will result in an error: the platform shall return RC_DENIED if the application calls acquire() and another interface has already been acquired.

The CUSS Technical Specification does not address the integration between the platform and the SBD device. Every common use platform that needs to control a self bag drop device will need to review the capabilities and command protocols for that SBD device and determine how to implement the CUSS interface and convert it into SBD native device commands.

The CUSS Technical Specification does not address the integration between SBD devices and airport baggage systems. Every airport that acquires and deploys self bag drop devices will need to integrate that SBD with the existing airport baggage and belt infrastructure. This integration might require PLC and other systems programming, custom wiring and messaging, and similar changes. None of these are in scope of the CUSS Technical Specification.

Description of Device:

Integrated Baggage Systems are complex devices allowing passengers to check-in their baggage themselves. Typically these devices are made from a set of separate devices for weighing and checking dimensions of the introduced bags as well as validating printed and attached baggage tags before these bags are fed into the airports baggage sortation systems. Scanning and validating baggage tags are based on the license plate definitions defined in the IATA Resolution 740 and may be also supported by RFID antennas.

Integrated Baggage Conveyors systems are not intended to X-ray baggage for explosives or other security critical items as this task is usually done prior or after the baggage check-in process.

To better reflect the process of baggage check-in, comprising of insertion and weighing, verification and waiting for a free slot on the carry-off belt the definition of the Integrated Baggage System always has three conveyor segments (InsertionBelt, VerificationBelt and ParkingBelt), even when there’s no physical representation of e.g. a verification belt.

Virtual Component Linking Diagrams:

A CUSS application that chooses to use the CUSS-SBD interface shall only acquire() the Conveyor, DataInput and DataOutput components listed below, and shall not acquire the component for AEA-SBD (UserOutput.)

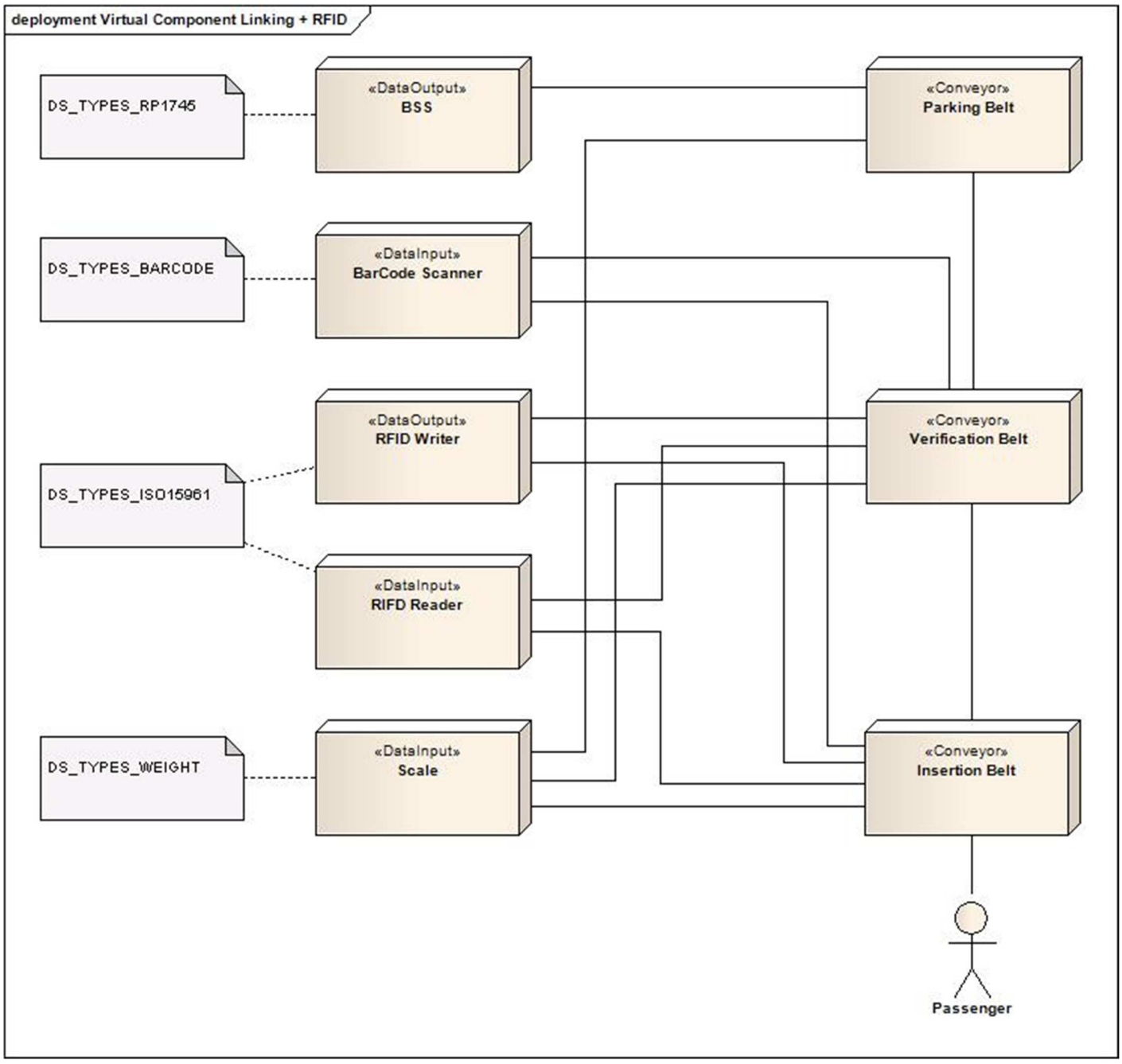


Figure 24: Linking for Integrated Baggage System including RFID support.

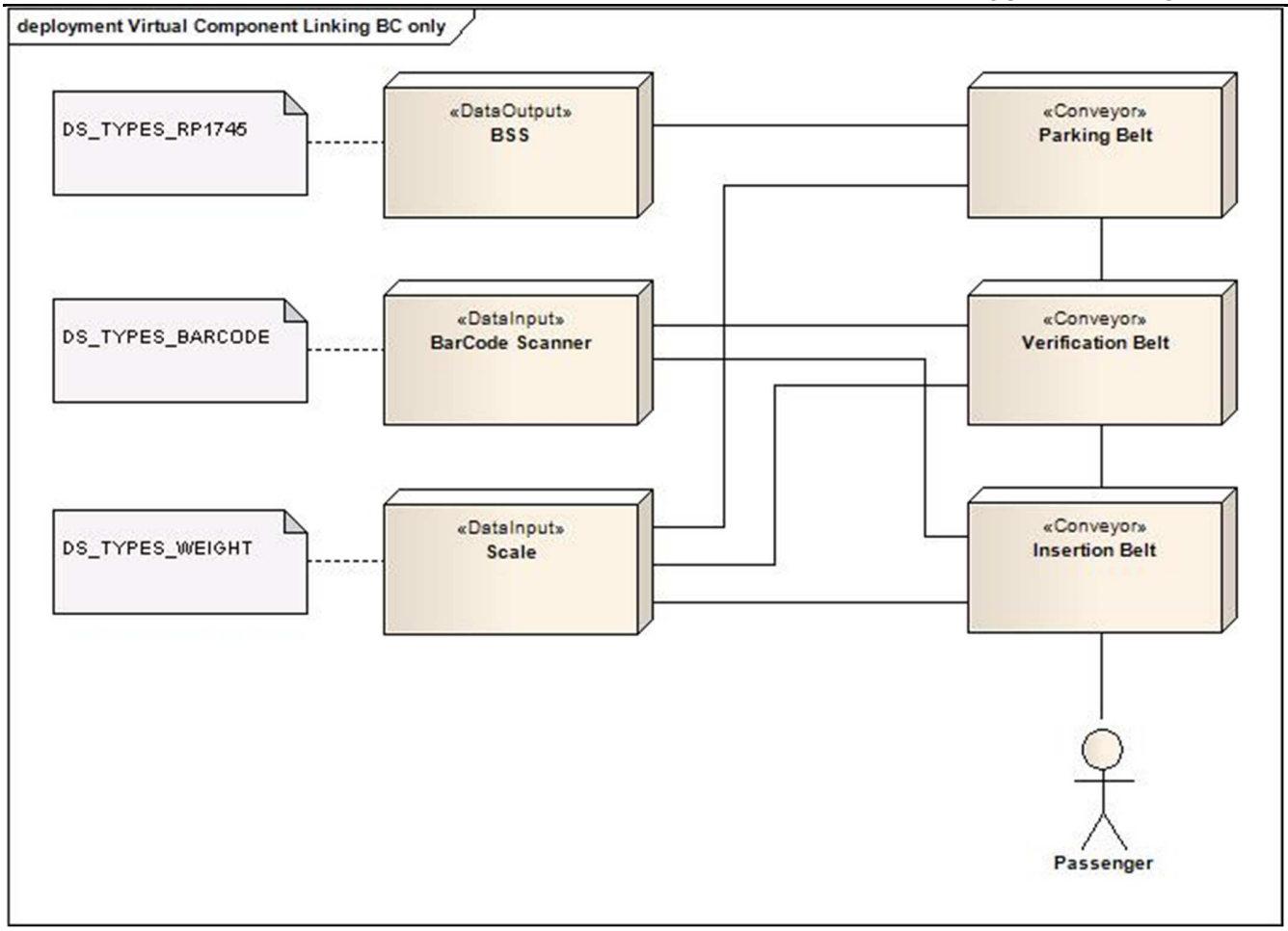


Figure 25: Linking for an Integrated Baggage System without RFID support.

Insertion Belt

The insertion belt is that part of the whole conveyor system that is connected to the passenger (Conveyor + User). It typically is connected to a scale for weighing and may also be connected to an RFID antenna for reading encoded RFID baggage tags (IATA standard encoding). The insertion belt provides an offer() directive allowing applications to wait for removal of a bag by the passenger.

The insertion belt allows moving baggage in forward direction only.

The virtual component will also include in the firmware version of its characteristics the string “DS_TYPES_SBDCUSS”.

Verification Belt

The verification belt describes the position on the conveyor where the weight is checked again to prevent fraud, where the printed and attached baggage tags are scanned or where encoded data on the RFID chips is read for verification.

The verification belt allows moving baggage in both directions forward and backward.

The virtual component will also include in the firmware version of its characteristics the string “DS_TYPES_SBDCUSS”.

Parking Belt

The parking belt allows parking/delaying a bag before feeding it into the Baggage Sortation System (BSS) of the airport. Parking a bag allows passengers to already continue the baggage check-in process with the next bag while the preceding bag is about to be moved on to the carry-off belt. It also allows applications to return bags to the passenger in case the baggage check-in transaction is cancelled or interrupted.

In terms of implementation, the parking belt is a derivation of the conveyor component implementation. Differently from the other conveyor implementations the parking belt may NOT allow the backward() command if the current parked bag is already in responsibility of the airport (e.g. BSM for that bag has been sent to the BSS but the bag is still waiting for a free slot on the carry-off-belt). Once a bag is physically forwarded onto the carry-off-belt the parking belt sends a BAGGAGE_ABSENT event to the application.

The virtual component will also include in the firmware version of its characteristics the string “DS_TYPES_SBDCUSS”.

Scale

Scale is a definition for a component that allowing to weigh baggage.

The scale component is implemented as a CUSS DataInput component. The scale component shall report only stable weights as data to the application. However, the platform may take into account unstable weights to detect and report if bags are present or not.

The virtual component can be identified by checking the firmware version of its characteristics for the string “DS_TYPES_WEIGHT”.

RFID

The RFID component represents one or more RFID antennas capable of reading and writing (encoding) RFID chips. The antenna that writes data to RFID baggage tags may be physically located within a baggage tag printer.

The RFID component is implemented as a CUSS DataInput and CUSS DataOutput component.

The virtual component can be identified by checking the firmware version of its characteristics for the string “DS_TYPES_ISO15961”.

Barcode Scanner

A barcode scanner component for reading bar-coded baggage tags as per IATA Resolution 740. The barcode scanner may also be linked to and physically located at the Insertion Belt.

The barcode scanner component is implemented as a CUSS DataInput component.

The virtual component can be identified by checking the firmware version of its characteristics for the string “DS_TYPES_BARCODE”.

BSS

The BSS component defines a standard CUSS DataOutput interface to the airports baggage sortation system. It allows passing Baggage Source Messages (BSM) to the airports sortation systems without knowing the details of the appropriate interfaces. This component is *optional.*

In the case that airports require flight data like flight numbers and -dates before baggage can be fed into the baggage sortation system these data elements then can be obtained from the provided BSM.

If the airport doesn't need to be informed about flight relevant data for a bag to be checked in and if there's no need to send BSMs to the airports sortation system the component will not be seen in the list of linked components.

The virtual component can be identified by checking the firmware version of its characteristics for the string "DS_TYPES_RP1745".

Distinct Characteristics:

Conveyor	
Characteristic	**Value**
maxWeight	The maximum weight of the baggage (in grams)
maxWidth	The maximum width of baggage (in millimeters)
maxHeigth	The maximum height of baggage (in millimeters)
maxLength	The maximum length of baggage (in millimeters)
maxBags	The maximum number of bags a conveyor can handle
onewayForward	If true, conveyor can only move into forward direction (the backward directive is not supported)
userInterferenceCapable	If true, conveyor system can detect and report user interference events which are not considered critical to health, safety and security
safetyIntrusionCapable	If true, conveyor system can detect and report safety intrusions which *are* considered critical to health, safety and security
barrierCapable	Indicates if the belt component has a physical barrier control controlling access to the belt insertion point.

Status Codes Definitions

Review Section 7.17.3 for more information on any of these status codes.

Normal Status:

Code	**Meaning**
BAGGAGE_ABSENT	No baggage present on position
BAGGAGE_FULL	Max. number of bags reached in this conveyor position
BAGGAGE_TRANSPORT_BUSY	Bag cannot be transported further at the moment (usually from parking position to the carry off belt)
BAGGAGE_PRESENT	Baggage present on position

Errors:

Code	**Meaning**
BAGGAGE_UNEXPECTED_CHANGE	The conveyor detected a change in the bag compared to a previous state (based on any factors that the Conveyor technology is able to detect that is not reported as dedicated data on another component)
BAGGAGE_INTERFERENCE_USER	A user interference (non-critical) was detected at some point on the conveyor. Operations may proceed.
BAGGAGE_OVERSIZED	Bag is too long/high/flat/short/heavy
BAGGAGE_TOO_FLAT	Bag is too flat

BAGGAGE_TOO_HIGH	Bag is too high
BAGGAGE_TOO_LONG	Bag is too long
BAGGAGE_TOO_MANY_BAGS	Baggage present, detected more than one baggage
BAGGAGE_TOO_SHORT	Bag is too short
BAGGAGE_WEIGHT_OUT_OF_RANGE	Weight exceeds weight range of conveyor system or SBD.

Fatal errors (These codes normally lead to system outage):

Code	Meaning
BAGGAGE_EMERGENCY_STOP	Person pressed the Emergency-Stop button.
BAGGAGE_INTRUSION_SAFETY	A critical security/safety intrusion was detected at some point on the conveyor. Operations are likely halted and in error condition until a supervisor reset.
BAGGAGE_JAMMED	Bag is jammed on conveyor
BAGGAGE_RESTLESS	Bag is permanently moving, maybe it is- or contains a living creature
BAGGAGE_MISTRACKED	The movement of a bag did not take place as expected when activating conveyors.
BAGGAGE_UNDETECTED	Unexpected baggage absent
BAGGAGE_UNEXPECTED_BAG	Unexpected baggage present.

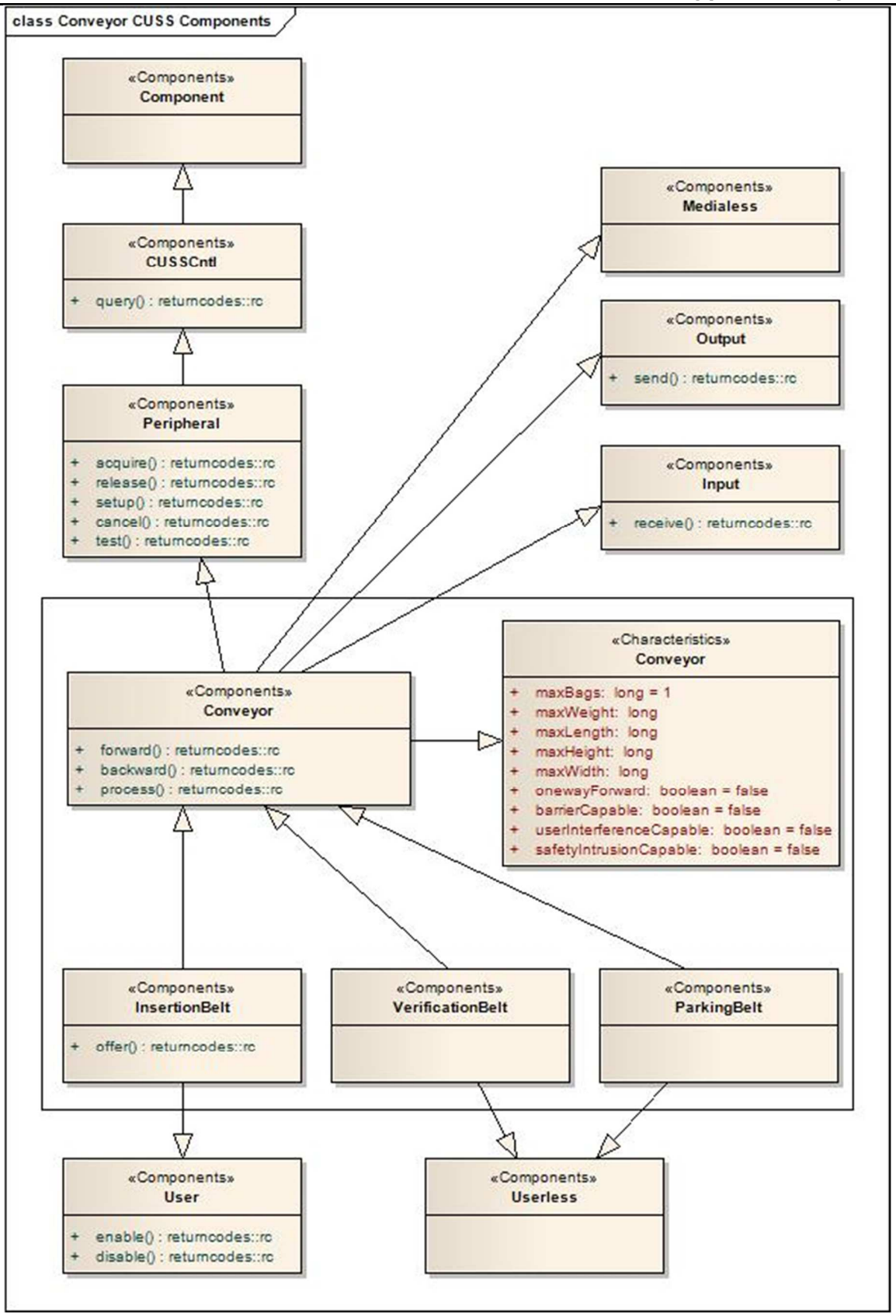


Figure 26: The class diagram for an Integrated Baggage System with directives per class.

7.17.1 Device Component Interface Directives Extension

This chapter defines the relation between new status codes and the conveyor virtual component directives. The relations of the standard CUSS status codes remain as they are defined in chapter [*Device Component Interface (DCI) Directives*].

7.17.1.1 acquire

Function: **acquire**	**Virtual Component Types**		
Status Code	InsertionBelt	VerificationBelt	ParkingBelt
BAGGAGE_ABSENT	X	X	X
BAGGAGE_FULL	X	X	X
BAGGAGE_UNEXPECTED_CHANGE			
BAGGAGE_WEIGHT_OUT_OF_RANGE			
BAGGAGE_INVALID_DATA			
BAGGAGE_OVERSIZED	X	X	X
BAGGAGE_PRESENT	X	X	X
BAGGAGE_TOO_FLAT	X	X	X
BAGGAGE_TOO_HIGH	X	X	X
BAGGAGE_TOO_LONG	X	X	X
BAGGAGE_TOO_MANY_BAGS	X	X	X
BAGGAGE_TOO_SHORT	X	X	X
BAGGAGE_MISTRACKED			
BAGGAGE_TRANSPORT_BUSY			
BAGGAGE_UNDETECTED	X	X	X
BAGGAGE_UNEXPECTED_BAG	X	X	X
BAGGAGE_JAMMED		X	X
BAGGAGE_EMERGENCY_STOP	X	X	X
BAGGAGE_RESTLESS	X	X	X
BAGGAGE_INTERFERENCE_USER	X	X	X
BAGGAGE_INTRUSION_SAFETY	X	X	X

The platform shall return RC_DENIED when an application attemps to call acquire() on a CUSS-SBD component after the application has already acquired a AEA-SBD component.

7.17.1.2 backward

The function allows moving a bag back to the previous position or back to the user.

If this directive is called on any Insertion component that has the onewayForward characteristic set, the platform shall return RC_NOT_SUPPORTED.

Note that the offer() directive should be used to return a bag to the user from the Insertion belt, not backward(). This is because the SBD device may have mechanical barriers and such that require special operation.

Function: **backward**	**Virtual Component Types**		
Status Code	InsertionBelt	VerificationBelt	ParkingBelt
BAGGAGE_ABSENT	X	X	X
BAGGAGE_FULL	X	X	X
BAGGAGE_UNEXPECTED_CHANGE			
BAGGAGE_WEIGHT_OUT_OF_RANGE			
BAGGAGE_INVALID_DATA			
BAGGAGE_OVERSIZED			
BAGGAGE_PRESENT	X	X	X
BAGGAGE_TOO_FLAT			
BAGGAGE_TOO_HIGH			
BAGGAGE_TOO_LONG			
BAGGAGE_TOO_MANY_BAGS			
BAGGAGE_TOO_SHORT			
BAGGAGE_MISTRACKED	X	X	X
BAGGAGE_TRANSPORT_BUSY			X
BAGGAGE_UNDETECTED	X	X	X
BAGGAGE_UNEXPECTED_BAG	X	X	X
BAGGAGE_JAMMED		X	X
BAGGAGE_EMERGENCY_STOP	X	X	X
BAGGAGE_RESTLESS	X	X	X
BAGGAGE_INTERFERENCE_USER	X	X	X
BAGGAGE_INTRUSION_SAFETY	X	X	X

7.17.1.3 cancel

This directive is a specific request that an operation in progress on the particular component be cancelled, if possible. For example, an application can attempt to cancel the forward(), backward(), offer(), send() and receive() directives.

Depending on the capabilities of the SBD components, and when the cancel request is made, there is no guarantee that the cancel request will be honoured. An application should track the condition of the affected components() after a cancel is complete.

The cancel() directive is NOT a request to halt a bag drop transaction completely. To halt a transaction completely, an application must implement the business logic required to control each component, return bags to customers, wait for customers to retrieve their bag, and similar tasks.

The cancel() directive shall return the status codes listed in section 3.6.10.1.

7.17.1.4 disable

The function disables user bag deposit on the insertion conveyor, either by activating a physical barrier or by activating interference/detection sensors. It is only available on the InsertionBelt component.

Applications can review the barrierCapable characteristic to determine if a barrier is present. However applications must call disable() even for devices without a physical barrier, as the platform and self bag drop device may need to perform other tasks to cease accepting bags.

Application suppliers should note that the decision of when to call enable() and disable() during a customer transaction is a business logic decision of the application in accordance with the above guidance.

Function: **disable**	**Virtual Component Types**
Status Code	InsertionBelt
BAGGAGE_ABSENT	X
BAGGAGE_FULL	X
BAGGAGE_UNEXPECTED_CHANGE	X
BAGGAGE_INVALID_DATA	
BAGGAGE_OVERSIZED	X
BAGGAGE_PRESENT	X
BAGGAGE_TOO_FLAT	X
BAGGAGE_TOO_HIGH	X
BAGGAGE_TOO_LONG	X
BAGGAGE_TOO_MANY_BAGS	X
BAGGAGE_TOO_SHORT	X
BAGGAGE_MISTRACKED	
BAGGAGE_TRANSPORT_BUSY	
BAGGAGE_UNDETECTED	X
BAGGAGE_UNEXPECTED_BAG	X
BAGGAGE_JAMMED	X
BAGGAGE_EMERGENCY_STOP	X
BAGGAGE_RESTLESS	X
BAGGAGE_INTERFERENCE_USER	X
BAGGAGE_INTRUSION_SAFETY	X

7.17.1.5 enable

The function enables user bag deposit on the insertion conveyor, either by removing a physical barrier or by suspending interference/detection sensors. It is only available on the InsertionBelt component.

Applications can review the barrierCapable characteristic to determine if a barrier is present. However applications must call enable() even for devices without a physical barrier, as the platform and self bag drop device may need to perform other tasks to prepare for bag acceptance.

Application suppliers should note that the decision of when to call enable() and disable() during a customer transaction is a business logic decision of the application in accordance with the above guidance.

Function: **enable**	**Virtual Component**
Status Code	InsertionBelt
BAGGAGE_ABSENT	X
BAGGAGE_FULL	X
BAGGAGE_UNEXPECTED_CHANGE	
BAGGAGE_WEIGHT_OUT_OF_RANGE	
BAGGAGE_INVALID_DATA	X
BAGGAGE_OVERSIZED	X
BAGGAGE_PRESENT	X
BAGGAGE_TOO_FLAT	X
BAGGAGE_TOO_HIGH	X
BAGGAGE_TOO_LONG	X
BAGGAGE_TOO_MANY_BAGS	X
BAGGAGE_TOO_SHORT	X
BAGGAGE_MISTRACKED	
BAGGAGE_TRANSPORT_BUSY	
BAGGAGE_UNDETECTED	X
BAGGAGE_UNEXPECTED_BAG	X
BAGGAGE_JAMMED	X
BAGGAGE_EMERGENCY_STOP	X
BAGGAGE_RESTLESS	X
BAGGAGE_INTERFERENCE_USER	X
BAGGAGE_INTRUSION_SAFETY	X

7.17.1.6 forward

The function allows moving a bag to the next position on the conveyor or to the airports take-away belt.

An application would typically verify the condition and capacity of the next belt in the chain (Insertion → Verification → Parking → Airport) prior to issuing this directive.

Important Note:

If the conveyor is the parking belt a call to forward() indicates an intent to promote the bag to the airport baggage belt and the bag is no longer in control of the SBD. Please review the description of

BAGGAGE_ACCEPTED and BAGGAGE_DELIVERED below for information on how to detect what happened to a bag forwarded from the parking belt.

Function: **forward**	**Virtual Component Types**		
Status Code	InsertionBelt	VerificationBelt	ParkingBelt
BAGGAGE_ABSENT	X	X	X
BAGGAGE_FULL	X	X	X
BAGGAGE_UNEXPECTED_CHANGE		X	X
BAGGAGE_WEIGHT_OUT_OF_RANGE	X	X	
BAGGAGE_INVALID_DATA			
BAGGAGE_OVERSIZED	X	X	X
BAGGAGE_PRESENT	X	X	X
BAGGAGE_TOO_FLAT	X	X	X
BAGGAGE_TOO_HIGH	X	X	X
BAGGAGE_TOO_LONG	X	X	X
BAGGAGE_TOO_MANY_BAGS	X	X	X
BAGGAGE_TOO_SHORT	X	X	X
BAGGAGE_MISTRACKED	X	X	X
BAGGAGE_TRANSPORT_BUSY			X
BAGGAGE_UNDETECTED	X	X	X
BAGGAGE_UNEXPECTED_BAG	X	X	X
BAGGAGE_JAMMED	X	X	X
BAGGAGE_EMERGENCY_STOP	X	X	X
BAGGAGE_RESTLESS	X	X	X
BAGGAGE_INTERFERENCE_USER	X	X	X
BAGGAGE_INTRUSION_SAFETY	X	X	X
BAGGAGE_ACCEPTED			X
BAGGAGE_DELIVERED			X

7.17.1.7 offer

The function allows waiting for a bag to be removed by the passenger, either by operating a physical barrier or by suspending interference/detection sensors. It is only available on the InsertionBelt component.

The directive waits for the customer to remove a bag from the insertion belt, then re-activates the physical barrier or interference/detection sensors.

Applications can review the barrierCapable characteristic to determine if a barrier is present. However applications should call offer() even for devices without a physical barrier.

Function: **offer**	**Virtual Component**

Status Code	InsertionBelt
BAGGAGE_ABSENT	X
BAGGAGE_FULL	X
BAGGAGE_UNEXPECTED_CHANGE	
BAGGAGE_INVALID_DATA	X
BAGGAGE_OVERSIZED	X
BAGGAGE_PRESENT	X
BAGGAGE_TOO_FLAT	X
BAGGAGE_TOO_HIGH	X
BAGGAGE_TOO_LONG	X
BAGGAGE_TOO_MANY_BAGS	X
BAGGAGE_TOO_SHORT	X
BAGGAGE_MISTRACKED	X
BAGGAGE_TRANSPORT_BUSY	
BAGGAGE_UNDETECTED	X
BAGGAGE_UNEXPECTED_BAG	X
BAGGAGE_JAMMED	X
BAGGAGE_EMERGENCY_STOP	X
BAGGAGE_RESTLESS	X
BAGGAGE_INTERFERENCE_USER	X
BAGGAGE_INTRUSION_SAFETY	X

7.17.1.8 process

The directive allows the application to request that the SBD examine and process the bag currently on a belt component in order to get data about the bag (dimensions, scan, RFID, and similar.)

In response, the platform will activate the bag detection capabilities on the SBD (which may involve belt movement and similar, and vary depending on the type of device), and report the current data back to the application for verification using DATA_PRESENT event or other events as applicable (such as BAGGAGE_TOO_LONG.)

If a Belt component does not support examination and processing of a bag and cannot return data for the bag, the platform shall return RC_NOT_SUPPORTED.

Function: **process**	**Virtual Component Types**		
Status Code	InsertionBelt	VerificationBelt	ParkingBelt
BAGGAGE_ABSENT	X	X	X
BAGGAGE_FULL	X	X	X

BAGGAGE_UNEXPECTED_CHANGE	X	X	X
BAGGAGE_WEIGHT_OUT_OF_RANGE	X	X	
BAGGAGE_INVALID_DATA	X	X	X
BAGGAGE_OVERSIZED	X	X	
BAGGAGE_PRESENT	X	X	X
BAGGAGE_TOO_FLAT	X	X	
BAGGAGE_TOO_HIGH	X	X	X
BAGGAGE_TOO_LONG	X	X	X
BAGGAGE_TOO_MANY_BAGS	X	X	X
BAGGAGE_TOO_SHORT	X	X	X
BAGGAGE_MISTRACKED	X	X	X
BAGGAGE_TRANSPORT_BUSY			X
BAGGAGE_UNDETECTED	X	X	X
BAGGAGE_UNEXPECTED_BAG	X	X	X
BAGGAGE_JAMMED	X	X	X
BAGGAGE_EMERGENCY_STOP	X	X	X
BAGGAGE_RESTLESS	X	X	X
BAGGAGE_INTERFERENCE_USER	X	X	X
BAGGAGE_INTRUSION_SAFETY	X	X	X

7.17.1.9 query

Function: **query**	**Virtual Component Types**		
Status Code	InsertionBelt	VerificationBelt	ParkingBelt
BAGGAGE_ABSENT	X	X	X
BAGGAGE_FULL	X	X	X
BAGGAGE_UNEXPECTED_CHANGE		X	X
BAGGAGE_WEIGHT_OUT_OF_RANGE	X	X	
BAGGAGE_INVALID_DATA			
BAGGAGE_OVERSIZED	X	X	X
BAGGAGE_PRESENT	X	X	X
BAGGAGE_TOO_FLAT	X	X	X
BAGGAGE_TOO_HIGH	X	X	X
BAGGAGE_TOO_LONG	X	X	X
BAGGAGE_TOO_MANY_BAGS	X	X	X
BAGGAGE_TOO_SHORT	X	X	X
BAGGAGE_MISTRACKED	X	X	X
BAGGAGE_TRANSPORT_BUSY			X
BAGGAGE_UNDETECTED	X	X	X
BAGGAGE_UNEXPECTED_BAG	X	X	X
BAGGAGE_JAMMED		X	X
BAGGAGE_EMERGENCY_STOP	X	X	X
BAGGAGE_RESTLESS	X	X	X
BAGGAGE_INTERFERENCE_USER	X	X	X
BAGGAGE_INTRUSION_SAFETY	X	X	X

The CUSS platform shall respond BAGGAGE_ABSENT instead of OK to query() requests when no bag is present and no other error condition is detected.

Notwithstanding the above requirement for CUSS platforms, for compatibility reasons it is recommended that CUSS applications should accept and interpret OK and BAGGAGE_ABSENT as equivalent.

7.17.1.1 receive

The receive() directive is used to return available bag dimension data from the specified conveyor. This data is included in accordance with the CUSS.SBD.XSD schema format.

7.17.1.2 release

Function: **release**	**Virtual Component Types**		
Status Code	InsertionBelt	VerificationBelt	ParkingBelt
BAGGAGE_ABSENT	X	X	X
BAGGAGE_FULL	X	X	X
BAGGAGE_UNEXPECTED_CHANGE		X	X
BAGGAGE_WEIGHT_OUT_OF_RANGE			
BAGGAGE_INVALID_DATA			
BAGGAGE_OVERSIZED	X	X	X
BAGGAGE_PRESENT	X	X	X
BAGGAGE_TOO_FLAT	X	X	X
BAGGAGE_TOO_HIGH	X	X	X
BAGGAGE_TOO_LONG	X	X	X
BAGGAGE_TOO_MANY_BAGS	X	X	X
BAGGAGE_TOO_SHORT	X	X	X
BAGGAGE_MISTRACKED	X	X	X
BAGGAGE_TRANSPORT_BUSY			X
BAGGAGE_UNDETECTED	X	X	X
BAGGAGE_UNEXPECTED_BAG	X	X	X
BAGGAGE_JAMMED	X	X	X
BAGGAGE_EMERGENCY_STOP	X	X	X
BAGGAGE_RESTLESS	X	X	X
BAGGAGE_INTERFERENCE_USER	X	X	X
BAGGAGE_INTRUSION_SAFETY	X	X	X

7.17.1.3 send

The send() directive is available on certain CUSS-SBD components but is reserved for future use.

7.17.1.4 setup

Function: **setup**	**Virtual Component Types**

Status Code	InsertionBelt	VerificationBelt	ParkingBelt
BAGGAGE_ABSENT	X	X	X
BAGGAGE_FULL	X	X	X
BAGGAGE_UNEXPECTED_CHANGE		X	X
BAGGAGE_WEIGHT_OUT_OF_RANGE	X	X	
BAGGAGE_INVALID_DATA			
BAGGAGE_OVERSIZED	X	X	X
BAGGAGE_PRESENT	X	X	X
BAGGAGE_TOO_FLAT	X	X	X
BAGGAGE_TOO_HIGH	X	X	X
BAGGAGE_TOO_LONG	X	X	X
BAGGAGE_TOO_MANY_BAGS	X	X	X
BAGGAGE_TOO_SHORT	X	X	X
BAGGAGE_MISTRACKED	X	X	X
BAGGAGE_TRANSPORT_BUSY			X
BAGGAGE_UNDETECTED	X	X	X
BAGGAGE_UNEXPECTED_BAG	X	X	X
BAGGAGE_JAMMED	X	X	X
BAGGAGE_EMERGENCY_STOP	X	X	X
BAGGAGE_RESTLESS	X	X	X
BAGGAGE_INTERFERENCE_USER	X	X	X
BAGGAGE_INTRUSION_SAFETY	X	X	X

7.17.1.5 test

Function: **test**	**Virtual Component Types**		
Status Code	InsertionBelt	VerificationBelt	ParkingBelt
BAGGAGE_ABSENT	X	X	X
BAGGAGE_FULL	X	X	X
BAGGAGE_UNEXPECTED_CHANGE		X	X
BAGGAGE_WEIGHT_OUT_OF_RANGE	X	X	
BAGGAGE_INVALID_DATA			
BAGGAGE_OVERSIZED	X	X	X
BAGGAGE_PRESENT	X	X	X
BAGGAGE_TOO_FLAT	X	X	X
BAGGAGE_TOO_HIGH	X	X	X
BAGGAGE_TOO_LONG	X	X	X
BAGGAGE_TOO_MANY_BAGS	X	X	X
BAGGAGE_TOO_SHORT	X	X	X
BAGGAGE_MISTRACKED	X	X	X
BAGGAGE_TRANSPORT_BUSY			X
BAGGAGE_UNDETECTED	X	X	X

BAGGAGE_UNEXPECTED_BAG	X	X	X
BAGGAGE_JAMMED	X	X	X
BAGGAGE_EMERGENCY_STOP	X	X	X
BAGGAGE_RESTLESS	X	X	X
BAGGAGE_INTERFERENCE_USER	X	X	X
BAGGAGE_INTRUSION_SAFETY	X	X	X

7.17.2 Data Formats

The following sections describe the data formats to be received or sent to the components of the baggage system. The basic data format for all data shall be a string allowing also the transmission of XML formatted data structures.

7.17.2.1 Bar-Code Scanner (DataInput)

The DataInput component for the bar-code scanner delivers its data in a CUSS msgDataType with the appropriate number dataRecords in it. The number of dataRecords reflects the number of barcodes scanned. It's the applications responsibility to validate and check the delivered data, especially when the msgDataType holds more than a single barcode or hold the correct barcode types.

It is anticipated that most SBD processing will use scanned License Plate Numbers as per IATA Resolution 740b, and that some SBD equipment may be optimized for or restricted to LPN barcodes. However the CUSS specification itself supports any type of barcodes including 2D, BCBP, and non-LPN barcodes.

Format specification (for LPN): **[0-9]{10}**
Example: 5220100478

7.17.2.2 RFID Scanner (DataInput)

The DataInput component for the RFID scanner delivers its data in XML format in a CUSS msgDataType. - It is ensured that only data from IATA Res.1740c compliant RFID tags is transferred to the application. If more than only one RFID tag is scanned, the CUSS application is responsible for handling the correct RFID tag. Therefore all RFID tags have a distinct tag identifier.

- The id is used to reference a specific/distinct RFID tag for encoding if the system detected more than one RFID tag on a bag. So, if the system detects more than one RFID (e.g. LH and QF permanent tags) on one bag it creates a unique reference for each tag found.
- For encoding (see below) the application will use the 'id' to address a specific tag.
- The 'key' describes a single tag object. Objects could be either IATA-1740c defined objects, e.g. the key for a LPN would be "1", baggage routing would be using the key "5". See below for examples, however please refer to the official IATA Passenger Services Conference Resolutions Manual for details.
- Another (specific) use case is to encode the LPN for the RFID tag directly from a LPN scanned from an attached bar-code. Therefore the application should set only: <property key="1" /> without any LPN data. - The SBD system then reads the LPN from an attached barcode (e.g. HPBT) and encodes the RFID chip together with the regular data (itinerary etc.) and the scanned LPN.
- Additional non-IATA "key" values can also be used, in accordance with this table:

Key	Description for Reading RFID

TAG	Whole content of memory bank 01, in Hex-ASCII. To transmit non-IATA-conform read results. I.e. special encodings.
TID	Whole content of memory bank 10, in Hex-ASCII. Used to identify RFID tag supplier.
APW	Content of memory bank 00, bit 32-63, in Hex-ASCII stored in element <property><binary> In case the access password is not readable (wrong access password or password is permanently locked) this identifier is not added to the data field.
KPW	Content of memory bank 00, bit 00-31, Hex-ASCII. In case the kill password is not readable (wrong access password or password is permanently locked) this identifier is not added to the data field.
KILL	Kill or make undetectable the RFID tag specified with the 'id' attribute. A kill password needs to be provided by the SBD application if the RFID tag is not from a baggage tag stock provided by the CUSS platform (printed locally/on demand). The kill password must be provided as Hex-ASCII in a separate "KPW" property.

IATA Passenger Services Conference Resolutions Manual

Table 1: Object Identifiers & Data Elements

Object Identifier	Object	Mandatory	Status	Memory Bank	Decoded Data Characters
1 0 15961 12 1	License Plate Code (10 digit)	Y	OTP	01	f[10]
1 0 15961 12 2	Flight date	C	OTP	01	m[3] See encoding rules below.
1 0 15961 12 3	Security Information	N	R/W	11	Selectee or not; Level of screening — 0 — 5 Screening cleared or failed See encoding rules below.
1 0 15961 12 4	Issuing Station	N	R/W	11	m[3]
1 0 15961 12 5	Baggage Routing	N	R/W	11	m[6-18]
1 0 15961 12 6	Flight Data	N	R/W	11	m[14-70]
1 0 15961 12 7	Passenger Name Data	N	R/W	11	m[2-26]
1 0 15961 12 8	Airline Frequent Flyer Level	N	R/W	11	f in the range 0 to 3
1 0 15961 12 9	Screening airport code	N	R/W	11	m[3]
1 0 15961 12 10	Destination Code	N	R/W	11	m[3]
1 0 15961 12 90	Door-to-door Delivery Service: Issue Date	N	R/W	11	f[4]
1 0 15961 12 91	Door-to-door Delivery Service: Sequence Number	N	R/W	11	3 numeric — f[3]
1 0 15961 12 92	EDS Processing for Door-to-door Delivery Service	N	R/W	11	12 alphanumeric — m[12] see encoding rules below.
1 0 15961 12 93	Door-to-door Delivery Service: Collecting Company	N	R/W	11	4 alphanumeric — m[4]
1 0 15961 12 94	Door-to-door Delivery Service: Invoice	N	R/W	11	15 numeric — f[15]
1 0 15961 12 95	Door-to-door Delivery Service: Delivering Company	N	R/W	11	4 alphabetic — m[4]
1 0 15961 12 127	Optional Data	N	R/W	11	Variable - m[n-m]

Format specification and example data: **msgDataType.records[0] XML message [CUSS.SBD.XSD]**

```
<?xml version="1.0" encoding="UTF-8"?>
<baggageData xmlns="urn:CUSS-1.3/types/conveyorInterface"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <rfidTagList>
            <rfidTag id="ID42">
                  <property key="1">
                        <string>5220100478</string>
                  </property>
                  <property key="2">
                        <string>55</string>
                  </property>
                  <property key="5">
                        <string>FRASIN</string>
                  </property>
            </rfidTag>
            <rfidTag id="ID56">
                  <property key="1">
                        <string>5220100479</string>
                  </property>
                  <property key="2">
                        <string>55</string>
                  </property>
                  <property key="5">
                        <string>FRAORD</string>
                  </property>
            </rfidTag>
      </rfidTagList>
</baggageData>
```

7.17.2.3 RFID Scanner (DataOutput)

The DataOutput component for the RFID Encoder receives its data in XML format in a CUSS msgDataType. The application uses the distinct tag identifier to address the right RFID tag for encoding.

XML data formatting is according to the XML schema mentioned below.

See the previous section for more information on the “id” and “key” values.

Format specification and example data: **msgDataType.records[0] XML message [CUSS.SBD.XSD]**

```
<?xml version="1.0" encoding="UTF-8"?>
<baggageData xmlns="urn:CUSS-1.3/types/conveyorInterface"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <rfidTagList>
            <rfidTag id="ID42">
                  <property key="1">
                        <string>5220100478</string>
```

```
                        </property>
                        <property key="2">
                              <string>55</string>
                        </property>
                        <property key="5">
                              <string>FRASIN</string>
                        </property>
                  </rfidTag>
            </rfidTagList>
</baggageData>
```

7.17.2.4 Scale (DataInput)

The DataInput component for the scale delivers its data in a CUSS msgDataType with a single dataRecord in it. Regardless of the reporting units or precision of the scale component, the platform must convert (if needed) and report the weight in metric grams. A weight of 0 grams is a valid weight.

Format specification and example data: **msgDataType.records[0] XML message [CUSS.SBD.XSD]**

```
<?xml version="1.0" encoding="UTF-8"?>
<baggageData xmlns="urn:CUSS-1.3/types/conveyorInterface"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <dimension>
            <load>
                  <weight>18000</weight>
                  <alibi>
                        <value>0123456789-ABCDEF</value>
                  </alibi>
                  <stable>true</stable>
            </load>
      </dimension>
</baggageData>
```

As part of commercial Weights & Measures regulations, some jurisdiction require that scales used for commercial transactions also provide a measurement tracking reconciliation number, sometimes called an alibi number, along with the measurement value.

If the scale and location require the use of alibi numbers, it is a platform requirement to provide the alibi reference to the application. If provided, the alibi number must be included as the second data record in the msgDataType array.

It is a platform implementation choice and task to determine how to generate, obtain, and track/audit weights and alibi numbers, and all other aspects of compliance with local or Weights & Measures requirements relating to alibi numbers.

It is an application business logic decision to detect and properly use alibi numbers provided by the CUSS platform. Usage requirements and alibi number syntax may vary from location to location, and cannot be described by the CUSS Technical Standard.

If a scale is able to detect unstable weight conditions, this condition shall be reported to the application as a BAGGAGE_PRESENT event, which will *not* include any weight value.

7.17.2.5 Dimensions (Insertion, Verification and ParkingBelt)

Because they are Input components, the InsertionBelt, VerificationBelt and ParkingBelt components may be able to provide extended bag dimension data as DATA_PRESENT events. They deliver this data in XML format in a CUSS msgDataType as record 0. XML data formatting is according to the XML schema mentioned below.

Format specification and example data: **msgDataType.records[0] XML message [CUSS.SBD.XSD]**

```
<?xml version="1.0" encoding="UTF-8"?>
<baggageData xmlns="urn:CUSS-1.3/types/conveyorInterface"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <dimension>
            <size>
                  <width>30</width>
                  <length>50</length>
                  <height>42</heigth>
                  <stable>true</stable>
            </size>
      </dimension>
</baggageData>
```

7.17.2.6 BSS (DataOutput)

The DataOutput component for the BSS accepts data in a CUSS msgDataType with a single dataRecord.

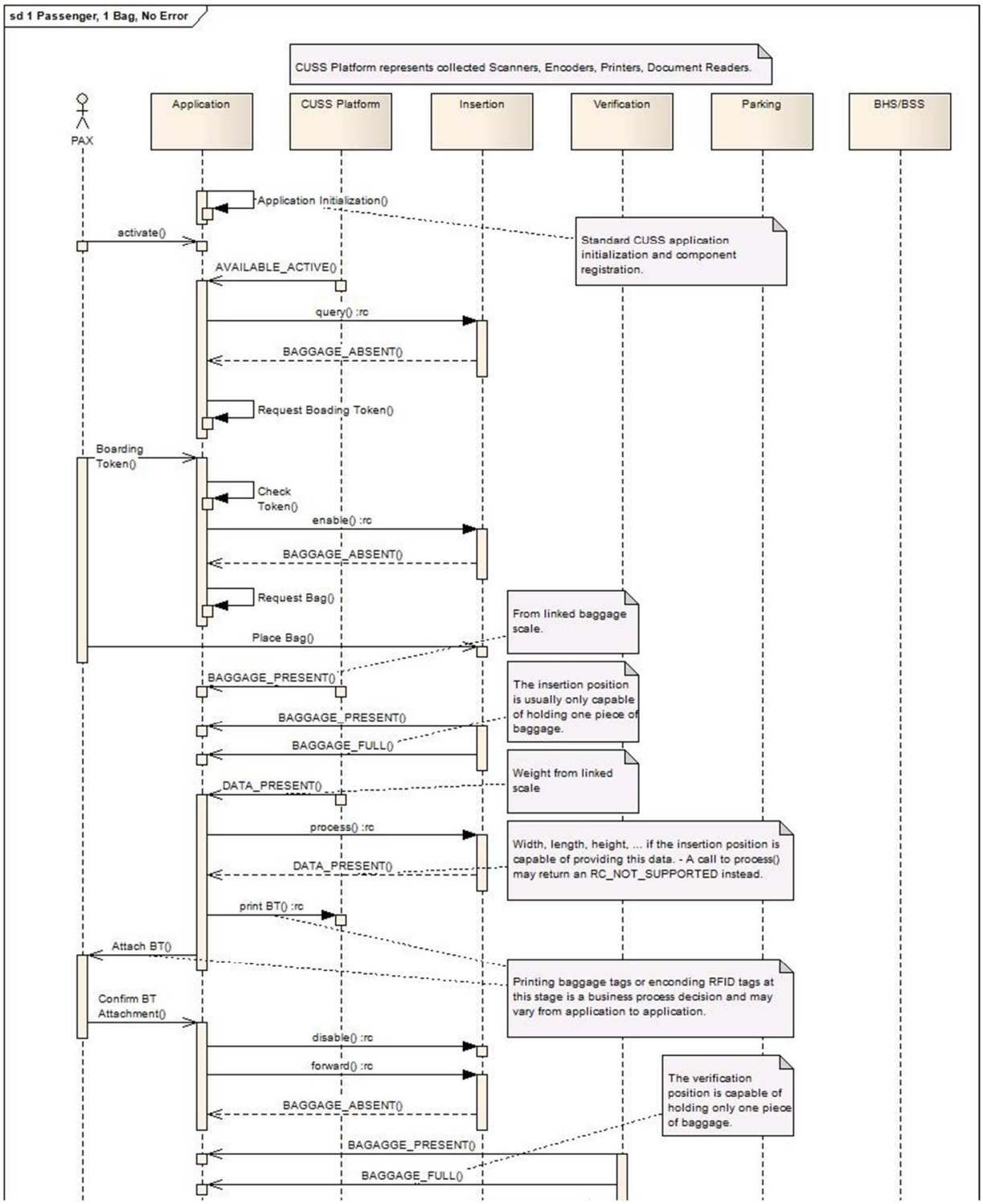
The data itself is a standard Baggage Source Message (IATA RP 1745) containing at least the following elements: **.V**, **.F** and **.N**.

Format specification and example data: **msgDataType.records[0] XML message [CUSS.SBD.XSD] and IATA Recommended Practice 1745 – Baggage Information**

```
<?xml version="1.0" encoding="UTF-8"?>
<baggageData xmlns="urn:CUSS-1.3/types/conveyorInterface"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <bsmMessage><![CDATA[BSM
.V1LFRA
.F/LH123/15MAR/BCN/F
.N0220567890001
.PGEHLING/AMR
ENDBSM]]>
      </bsmMessage>
</baggageData>
```

Typical Sequence Diagram for Integrated Baggage System:

In addition to the example show nere, review section 7.17.4 for further examples of how to control an SBD device under certain situations.



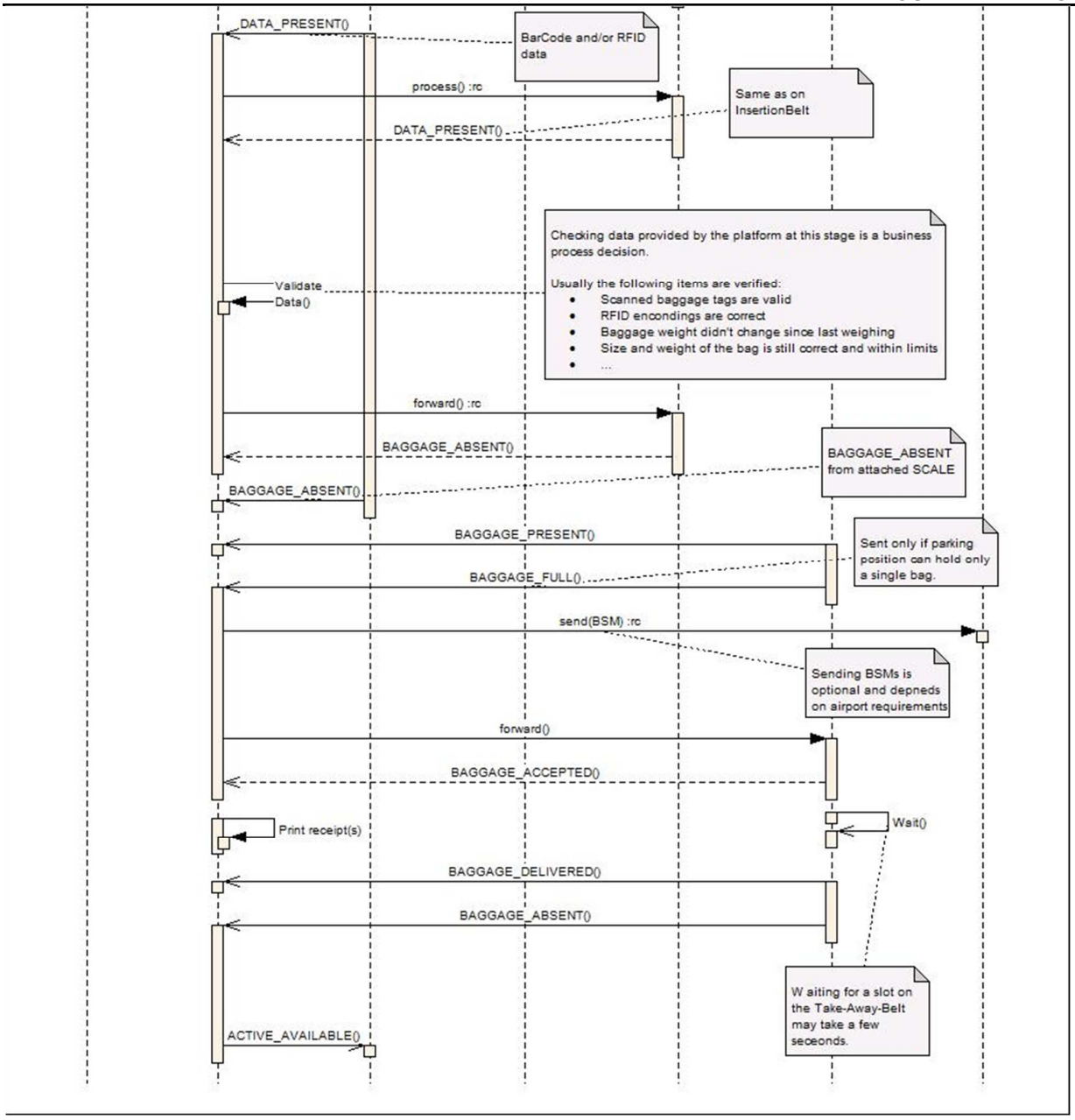


Figure 27: Checking-in one bag only.

7.17.3 Notes and Comments on Implementation

7.17.3.1 Deprecation of the existing CUSS 1.1 Conveyor interface

A previous version of the CUSS Technical Specification included a component model for self bag drop systems that consisted of a single Conveyor component. This approach was introduced in CUSS 1.1 in 2005.

Since then, numerous implementations of Baggage Scale and Self Bag Drop solutions using the CUSS 1.1 Conveyor definition were deployed successfully by multiple vendors. However, some of the deployments had problems understanding the Conveyor definition, or implementing required application functionality via the Conveyor component.

For this reasons, as part of updating the CUSS standard to 1.3, the technical participants in the CUSS Technical Solution Group decided to deprecate the existing Conveyor interface, and implement a new CUSS-SBD interface that is not backwards compatible with the Conveyor interface from CUSS 1.1.

Deprecation means that although an interface remains in the Technical Specification for compatibility with existing applications and platforms, the interface is intended to be removed from the next version of the standard. **For this reason CUSS applications should not rely on deprecated interfaces being available in future releases of the CUSS Technical Specification.**

The design goals of this new interface include:

- Extend the virtual component model from a single component to three components, to better reflect how bags are processed at a drop off point as well as the actual design of some Self Bag Drop devices.
- Eliminate any “black box” behavior desicisions in the component that implement business logic or bag processing rules that should be the application’s choice, not the platform’s
- Have a model that allows for a clear and well-defined implementations for baggage scales only (not attached to a conveyor mechanism)
- Eliminate redundant error codes that duplicated concepts that inherently exist in the base CUSS component model
- Allow more flexibility for changes for future Self Bag Drop needs that do not require a change to CUSS IDLs.
- Ensure that typical bag processing requirements can be accomplished using the CBS component model, and provide sample application usage flows demonstrating some of those cases
- Ensure that processing that should reside in the SBD and platform and PLC remain there, and are not imposed on the application.
- Support an airline industry adoption of AEA-SBD as a control protocol by offering an alternative self bag drop conveyor control interface. Alongside CUSS-SBD, for application suppliers that prefer a simpler command model.

The result of this process is the new three-Belt CUSS-SBD component model in this section.

While it unfortunately is not backwards compatible with the previous interface, and applications and platforms that use the previous interface will require updates to be deployed under CUSS-TS 1.3, the TSG anticipates that this new interface will be easier to use and more consistently implemented across platforms and SBD devices.

7.17.3.2 Weights and Dimensions, and Data Formats

This section provides information about the concepts of weights, dimensions, and limits for scales, conveyors, and airport baggage systems.

maxWeight, maxWidth and other dimension limit Characteristics

Each belt component includes a set of characteristics that indicate the dimension limits of bags that can be processed on the self bag drop device.

These weight, length, height and width limits represent the combined limits of the physical SBD unit itself, and the overall limits of the airport baggage system to which the SBD is connected.

An application should review these limits as part of its business logic for detecting and processing bags. The weight limit is in grams, and the dimension limits are in centimeters.

maxBags belt capacity Characteristic

Each belt component includes a characteristic that indicates the number if simultaneous bags that belt can process.

Depending on the capability of the device, a self bag drop conveyor will likely support only one one bag per belt position. Advanced SBD devices may support multiple bags in the Parking Belt position.

An application should review these bag capacity limits as part of its business logic for detecting and processing bags. The application may choose to exploit the full capability of a SBD that supports multiple simultaneous bags by implementating the advanced logic and monitoring required to operate correctly in this configuration.

CUSS platforms will correctly report BAGGAGE_ABSENT, BAGGAGE_PRESENT and BAGGAGE_FULL events on each belt position in view of the physical number of bags present on the belt, and the belt's maximum capacity. Applications should use these events to determine if room exists for processing on a particular belt, not the maxBags setting.

An SBD that does not include a distinct physical belt for the Insertion Belt, Verification Belt and Parking Belt, will share the same maxBags limit for each belt. For example, an SBD with a single belt

with a capacity of one bag will indicate maxBags=1 on three logical belt components, but will report the BAGGAGE_FULL events as required and will have a total capacity of one bag (not three.)

The send() directive

Each belt component includes a send() directive. This directive is reserved for future use and does not yet have a defined behavior.

The RFID Encoder component is a DataOutput component that supports send() directives. The data being sent must comply with the XML format mentioned elsewhere in this section. The XSD is also provided as interface file component of the CUSS Technical Specification.

The receive() directive

Each belt component includes a receive() directive, to be used to obtain data from the belt component. The data is reported in XML format in accordance with the XSD definition included with the CUSS Technical Specification.

The belt components return dimension data. Other data such as scanned barcodes are returned via the appropriate DataInput component (for example, the scale DataInput component returns weight values.)

The belt components will only provide data in response to a process() directive. If the belt does not have a capability of returning data, the platform will response RC_NOT_SUPPORTED when the application calls process().

The RFID Reader, Barcode Scanner, Weight Scale, and other DataInput components return their own data via the receive() directive. Those components will report the data in response to a process() directive on the linked Belt component. The data format will be as defined for each component (see above.)

Applications must combine and interpret the data obtained from different components in accordance with their internal business logic requirements. For example, it is an application business logic responsibility to obtain data from the Verification belt and determine if the bag has been changed or not.

Bar Code Scanning

Most self bag drop devices include barcode scanning capabilities as an essential component of accepting airline bags. This barcode scanning capability is separate and distinct from the barcode scanner used on the kiosks to read boarding passes or mobile phones.

The barcode scanner component of the SBD will indicate the DS_TYPES_BARCODE Characteristic.

If the scanner is capable of reading multiple barcodes at once, the CUSS platform will report multiple data message records to the application, one for each unique barcode value that is scanned.

It is a CUSS requirement that all SBD barcode scanners be able to read IATA Resolution 740 license plate numbers (LPNs) using the Interleaves 2 of 5 format. There is no require nor is there any restriction in this CUSS Technical Standard on devices that report additional types of barcodes, such as 2D barcodes read from permanent or home-printed bagtags.

Heavy Tag printing and data formatting

The CUSS Technical Specification does not define a standard Heavy Tag document size or layout. Please review Section 7.17.5 and refer to Appendix G for more information on Heavy Tag Printing.

7.17.3.3 Detecting and Notification of Bags

While self bag drop devices will include a scale component and the ability to read weight values, it is important to note that SBD devices do not necessarily use the scale weight measurements (exclusively, or at all) to detect if bags are present.

So while applications have access to the weight values directly from the scale components, applications should not be designed to assume that the weight is the only way to detect a bag.

In particular, the SBD device's ability to detect bags may or may not be based on weight readings from the scale. In addition to or instead of weight measurements, an SBD could use any volumetric, seismic, photographic, or other non-scale sensing capabilities can be used to detect bags.

BAGGAGE_ABSENT: No bag present on Belt or Scale

If the platform does not detect any bag on a Belt component, either by checking for weight or via any other method, it will report BAGGAGE_ABSENT.

For the Scale component of the self bag drop device, BAGGAGE_ABSENT indicates that the scale is stable and at the zero point (or within the zero-point range.)

BAGGAGE_ABSENT is a _public event that can be monitored even when an application is not ACTIVE.

BAGGAGE_PRESENT: Bag(s) detected on Belt

If the platform detects any bag on a Belt component, either by checking for weight or via any other method, it will report BAGGAGE_PRESENT.

If no futher bags can be added to a belt (in other words, the maxBags limit has been reached for a particular belt) the platform will then report BAGGAGE_FULL.

Because the Scale component is a DataInput, when the scale reads a stable, non-zero weight it will not report BAGGAGE_PRESENT but will send a DATA_PRESENT instead. A customer manipulating a bag may lead to a sequence of BAGGAGE_PRESENT and DATA_PRESENT events as the weight changes multiple times between unstable and stable.

Depending on the mechanical layout of the self bag drop conveyor belt/scale components, the BAGGAGE_PRESENT/BAGGAGE_ABSENT status of the scale component may or may not be in sync with the BAGGAGE_PRESENT/BAGGAGE_ABSENT statuses of the belt components.

Bag movement on the belts or customer interaction with the bag (such as attaching a tag) will likely generate unstable weights on the scale components. This is normal and should not be viewed as an error by applications.

Applications should use this event to get the weight from the bag(s) placed on the associated Belt components, but as mentioned above they should not rely exclusively on the scale weight indication to detect whether or not bags are present.

Applications should anticipate that at the start of that application's transaction, the insertion and verification belts will be empty but the parking belt *may* have a bag on it already from the previous transaction. This is a normal and acceptable condition.

BAGGAGE_PRESENT is a _public event that can be monitored even when an application is not ACTIVE. For belts with a single bag capacity BAGGAGE_PRESENT will be followed almost immediately by a BAGGAGE_FULL event.

BAGGAGE_FULL: Bag(s) detected on Belt and no more room on the belt

If the platform detects a bag on a Belt component, and there is no room (physically, or logically) on the belt for additional items, the platform will report BAGGAGE_FULL for the belt.

Applications must be written to support both the BAGGAGE_PRESENT and BAGGAGE_FULL events when monitoring the self bag drop device for bags. In particular, a large portion of SBD equipment will only support one bag per belt, which will immediately result in a BAGGAGE_FULL event as soon as a single bag is detected.

Applications should use the BAGGAGE_FULL event/status to determine which belt positions have capacity for more bags. In particular, a parking belt that is not full means additional bags can be processed on the insertion and verification belts while parked bags are waiting to be dispatched, even across transactions.

Applications should anticipate that at the start of that application's transaction, the insertion and verification belts will be empty but the parking belt *may* already be full due to bag(s) from the previous transaction. This is a normal and acceptable condition.

BAGGAGE_FULL is a _public event that can be monitored even when an application is not ACTIVE.

BAGGAGE_UNEXPECTED_CHANGE: bag change detected

Some self bag drop devices may include sensors and detection capabilities to detect when a bag has changed compared to the bag that was previously there. This detection capability is *in addition* to any ability to read and detect change in the bag weight, barcode(s), or RFID tags.

For example, a self-bag-drop device may include a camera and volumetric video analysis software to detect that the shape, position or color of a bag has changed.

If the SBD is able to detect a change in a bag that is not related to the bag's weight, barcodes, or RFID data (or any other data already provided to the application via a dedicated DataInput component) then the platform should report this condition as a BAGGAGE_UNEPECTED_CHANGE.

It is an application business logic decision whether to react to and if and how it processes this event. It is anticipated but not required that it would be treated similar to a user interference event or receiving inconsistent weight data from the scale component.

In addition to checking for BAGGAGE_UNEXPECTED_CHANGE, it is an application business logic responsibility to also monitor for changes in weight, barcodes, RFID information, by reading and comparing these measurement values received by reading directly from the Belt and DataInput components.

BAGGAGE_WEIGHT_OUT_OF_RANGE and similar: dimensions exceeded

Some self bag drop devices may include sensors and detection capabilities to detect when the dimensions of a bag exceed to system limits set for the SBD device and connected airport baggage system.

If these conditions can be detected, the platform should report them, in order of priority from highest to lowest, as:

- BAGGAGE_OVERSIZED - exceeds multiple dimensions
- BAGGAGE_TOO_MANY_BAGS - collection of items detected
- BAGGAGE_WEIGHT_OUT_OF_RANGE - outside of acceptable weight range
- BAGGAGE_TOO_HIGH - exceeds height limitation
- BAGGAGE_TOO_LONG - exceeds length limitation
- BAGGAGE_TOO_SHORT - does not meet length minimum requirements
- BAGGAGE_TOO_FLAT - does not meet height minimum requirements

It is an application business logic decision whether to react to and if and how it processes these events. It is anticipated but not required that bag processing will halt and the application will instruct the user on processing limitations for bags at this position.

If an application ignores these conditions and attempts to continue processing the bag using the process(), forward() or other directives, the platform and SBD may generate addition errors such as BAGGAGE_MISTRACKED or similar.

BAGGAGE_ACCEPTED and BAGGAGE_DELIVERED: bag transfer to the airport belt

An application issues a forward() directive on the Parking Belt to instruct the platform and self bag drop device to convey the furthest bag on the parking belt, onto the airport baggage system belt.

Depending on the design and capabilities of the airport and SBD belt systems, bags may physically remain on the parking belt even though ownership and control is now in the hands of the airport baggage system.

For example, the PLC on the parking belt may be waiting for an “open slot” on the airport belt to appear, at which point the bag will automatically be promoted to the airport belt without any input from the application or the platform.

When the forward() directive on the parking belt is successful, subject to the timeout parameter provided by the application, the platform shall respond BAGGAGE_ACCEPTED if the bag remains physically on the parking belt, and shall response BAGGAGE_DELIVERED if the bag is no longer on the parking belt. In both cases, the bag is under control of the airport baggage system and cannot be returned to the customer.

Neither BAGGAGE_ACCEPTED nor BAGGAGE_DELIVERED shall be responses to the query() directive.

BAGGAGE_DELIVERED shall be provided as a _private event notification once an accepted bag moves from the parking belt to the airport belt following a BAGGAGE_ACCEPTED response to the forward() directive.

It is an application business logic decision to determine how to monitor and process these events. The application may receive BAGGAGE_DELIVERED events even when it is no longer active.

However, generally speaking, an application should continue to process additional bags if needed, or end its CUSS session, even if an accepted bag has not yet reported as delivered. In other words, it is safe for an application to end its transaction even if a bag is still on the parking belt, as long as it is under the airport baggage system’s control (as indicated by a BAGGAGE_ACCEPTED response.)

BAGGAGE_TRANSPORT_BUSY: availability of the airport baggage belt

Some platforms and baggage systems may be able to detect if the airport baggage system is able to immediately accept bags (there is a physical or logical position available on the baggage belt.)

If a platform can detect when there are no immediate open slots available on the airport belt system, it shall report the BAGGAGE_TRANSPORT_BUSY condition on the Parking Belt. This condition does not apply to any other component of the SBD.

It is an application business logic decision to optionally check for this condition, and delay or modify the customer bag transaction in response to this event. Generally speaking, however, this event should not be used to prevent or restrict the progress of a transaction, except to provide an indication to the customer that even though their transaction is finished, their bag may be delayed on the parking belt.

7.17.3.1 Interference, Intrusion and Error Conditions

userInterferenceCapable and **BAGGAGE_INTERFERENCE_USER**

The three logical belt areas of an SBD may have the ability to to detect when a person or item intrudes into the belt area at a time when no activity is expected.

If the intrusion is not considered a safety or security hazard, it is considered a *user* interference. This includes areas that might be accessed casually by a traveler at the airport, such as reaching into a scale area or temporarily depositing an item on a surface.

> **The CUSS platform and/or SBD device *will not necessarily* (but *may*) halt baggage processing and belt or mechanical movement in response to a user interference event.**

Here are examples of events which would be considered user interference events (if they occur while the InsertionBelt is not enabled() and expecting customer interaction):

A light curtain or sensor is obstructed in the area around the insertion belt	Instruct the user to avoid reaching into the baggage area, then wait for the interference to stop and resume the transaction, reprocessing the bag.
Weight is detected on a scale component connected to the insertion belt	Confirm if they passenger wishes to proceed with the transaction prior to intiating any baggage movement, and then enable the belt and process as normal.
Someone attempts to lift a barrier or bar to gain access to the insertion belt	Instruct the user to cease interfering with the device, verify the condition of the belt and barriers, then restart or resume bag processing that had been in progress.
Thermal, imaging or seismic sensors detect someone reaching into the insertion or verification areas	Instruct the user that the bag cannot be modified once the process has started, wait for the interference to stop, then roll back and restart processing for that bag.
An object is thrown into the area triggering motion or image sensors	Instruct the user to cease interfering with the device, verify the condition of the belt and barriers, then restart or resume bag processing that had been in progress.

The exact user detection capabilities and methods for detecting interference and intrusion will vary from device to device. A CUSS platform that includes an SBD device will need to:

- Determine what user interference detection capabilities exist on the SBD (if any)
- Map that capability to one or more of the belt components in an appropriate fashion
- Insure the belt component characteristic userInterferenceCapable is correct set
- Correctly detect interference conditions, report them to the application, and report again when the intrusion is resolved by reverting the component back to the appropriate BAGGAGE status.

For the InsertionBelt component, the platform should only report interference conditions if the component is not enabled and is not performing an offer(), since those enable and offer requests imply and expect user interaction with the belt area.

As a user interference is not a safety or security hazard, the way in which an application handles the event should treat it as a non-critical event. The CUSS standard does not mandate any specific steps to take, as this is a business logic decision for the self bag drop application.

A likely behavior sequence for an application responding to BAGGAGE_INTERFERENCE_USER would be:

- Notify the user to avoid interfering with the machine
- Interrupt any bag processing that the intrusion may have interfered with (for example, weight verification)
- Monitor the component status until the intrusion condition is resolved, up to a certain timeout (for example, a minute)
- Do not assume anything about positions of bags after this event; instead, verify with status calls
- It is expected that most users will cease the intrusion and the transaction can proceed normally after a moment.
- Check the condition of the bags on all logical belt positions
- Resume the transaction if recovered within a specified period of time
- Cancel the transaction if there is no recovery in time

safetyIntrusionCapable and **BAGGAGE_INTRUSION_SAFETY**

The three logical belt areas of an SBD may have the ability to to detect when a person or item intrudes into a belt area that would be considered a safety or security hazard.

A safety or security condition is one where an item or person is in an area that must be vacant at all times except when processing bags and specifically directing bags into that area.

Any event that requires (based on local safety requirements) automatic and immediate mechanical stoppage or any sort must be reported by the platform as a safety intrusion event.

Any situation that requires intervention from airport or airline staff to verify the condition of the belt systems must be reported by the platform as a safety intrusion event.

The CUSS platform and/or SBD device *must* halt all belt or mechanical movement in response to a safety intrusion event, and comply with any additional local safety and security requirements in effect.

Here are examples of events which would be considered safety intrusion events (if they occur not as a result of an intended bag movement):

A bag movement is detected at the Verification or Parking belt that is not expected (such as caused by an animal or person crawling across the belts)	Ensure that application instructions or local signage indicate that access to the belt area is a safety concern and people should remain clear. Wait for up to 60 seconds for the situation to resolve before resuming, or go unavailable until the condition is resolved.
An object moves from the parking belt to the airport belt that is not expected	Ensure that application instructions or local signage indicate that bag processing begins at the insertion point and is then automated to move the bag forward and to accept the bag, and that passengers should only deposit the bag at the insertion point.
Thermal, imaging or seismic sensors detect someone reachine into the parking area	As above.

Restless bag or live bag detection (depending on local requirements)	Ensure that application instructions or local signage indicate that bag processing is only for cargo luggage and to avoid placing pets or children on the belt, and that pet carriers require processing at a desk.
An Emergency Stop is triggered elsewhere on the baggage processing system that affects the SBD position	As above.

The exact user detection capabilities and methods for detecting intrusion will vary from device to device. A CUSS platform that includes an SBD device will need to:

- Determine what safety intrusion detection capabilities exist on the SBD (if any)
- Map that capability to one or more of the Belt components in an appropriate fashion
- Insure the belt component characteristic safetyIntrusionCapable is correct set
- Correctly detect safety intrusion conditions, report them to the application
- Correctly detect when the intrusion condition is resolved (via manual intervention by staff or otherwise) and revert all SBD components back to the appropriate current status.

Any event that requires an immediate halt to bag processing or device movement for health, safety and security reasons must be reported as a safety intrustion. In addition, it is the responsibility of the platform and/or its devices (PLC, etc.) to immediately halt bag processing and device movement in this situation.

There is no requirement for CUSS applications using the device to issue interface requests to halt processing or movement and can assume that this hard stop has already taken place. After this point, the CUSS standard does not mandate any specific steps or business logic decisions that should take place in the self bag drop application.

A likely behavior sequence for an application responding to BAGGAGE_INTRUSION_SAFETY would be:

- Notify the user to avoid interfering with the machine
- Interrupt any bag processing that the intrusion may have interfered with (even though the SBD and platform shall do any mechanical stops needed for health, safety and security automatically)
- Indicate to the user that the transaction cannot proceed
- Wait some predetermined time to see if the transaction is resolved (staff intervention, etc.)
- Do not assume anything about positions of bags after this event; instead, verify with status calls
- If no resolution, end the session and go to UNAVAILABLE
- Monitor the component status until the intrusion condition is resolved
- Set the application back to AVAILABLE and allow new transactions to start

Emergency Stop and BAGGAGE_EMERGENCY_STOP

An emergency stop is a special case of a safety intrusion that merits specific handling as it is a near-universal characteristic of baggage handling systems and self bag drop devices. As such, the handling of an emergency stop event is similar to handling a safety intrusion event.

The CUSS platform and/or SBD device *must* halt all belt or mechanical movement in response to an emergency stop event, and comply with any additional local safety and security requirements in effect.

A CUSS platform that includes an SBD device will need to:

- Determine what the local Emergency Stop requirements are for the SBD and attached baggage belt system
- Correctly detect emergency stop conditions and
  - Take any action as per local requirements
  - Indicate the event to the CUSS application
  - Monitor the emergency stop condition until it is resolved (via manual intervention by staff or any other mandated local process)
- Once resolved revert all SBD components back to the appropriate current status.

The requirements for responding to an emergency stop event may vary from site to site. It likely requires an immediate halt to bag processing or device movement on the SBD, but may also require that all other SBDs, belts, or conveyors in the area also halt immediately. It is the responsibility of the platform and/or its devices (PLC, etc.) to immediately perform all halts for bag processing or device movement in response to an emergency stop event.

There is no requirement for CUSS applications using the device to issue interface requests to halt processing or movement and can assume that the platform and SBD have already done this. After this point, the CUSS standard does not mandate any specific steps or business logic decisions that should take place in the self bag drop application.

A likely behavior sequence for an application responding to BAGGAGE_EMERGENCY_STOP would be the same as for handling a BAGGAGE_SAFETY_INTRUSION event.

BAGGAGE_RESTLESS: Unstable bag detected on a belt

Some self bag drop devices may include sensors and detection capabilities to detect when bags are not stable. Unstable bags are considered restless bags, and include items or carriers containing pets, children, or items that shift or vibrate.

If a platform is capable of detecting this condition and if there are local requirements that restless bags be treated as a safety event, for example requiring that all belt movement be stopped, then the platform must treat a restless bag as a safety intrusion event, processing it and reporting it as a BAGGAGE_RESTLESS event.

It is an application business logic decision whether to react to and if and how it processes this event. It is anticipated but not required that it would be treated similar to a safety intrusion event.

BAGGAGE_MISTRACKED and similar: Mistracked or unexpected bags

A mistracked bag occurs when commanded belt movements take place without error, but the corresponding bag movement(s) do not complete as logically expected.

> **The CUSS platform and/or SBD device *must* halt all belt or mechanical movement in response to a mistracked bag event, and comply with any additional local safety and security requirements in effect.**

A CUSS platform that includes an SBD device will need to:

- Determine what the mistracked bag requirements are for the SBD and attached baggage belt system
- If mistracked bag detection is needed, then correctly detect mistracked bag conditions and:
  - Take any action as per local requirements
  - Indicate the event to the CUSS application
  - Monitor the mistracked condition until it is resolved (via manual intervention by staff or any other mandated local process)
- Once resolved revert all SBD components back to the appropriate current status.

All processing for BAGGAGE_MISTRACKED should be considered a safety intrustion, and all requirements for processing safety intrustions is as described above.

Likewise, the business logic behavior for an application responding to BAGGAGE_MISTRACKED would probably be very similar as for handling a BAGGAGE_SAFETY_INTRUSION event.

Two additional events are similar to a mistracked bag. Generally speaking, platforms should detect (if possible) and report these conditions, and applications should make a business logic decision to process the events, in a fashion similar to BAGGAGE_MISTRACKED.

- BAGGAGE_UNEXPECTED_BAG - bag appeared without corresponding tracking request
- BAGGAGE_UNEXPECTED_BAG - bag abandoned at the end of a transaction (see below)
- BAGGAGE_UNDETECTED - bag disappeared without corresponding tracking input

7.17.4 Abandonned Bag and Session Cleanup requirements

For self bag drop operations, there is a risk that an in progress transaction will result in an abandoned bag. For the purposes of the CUSS Technical Specification, an abandoned bag is:

> *Any bag that remains on or within the self bag drop device, when no CUSS airline application is in the ACTIVE state, and which has not been accepted into direct control of the airport baggage system , is considered an Abandoned Bag*.

At the end of every CUSS application transaction, the CUSS platform shall verify the self bag drop device for any abandoned bags, and shall report BAGGAGE_UNEXPECTED_BAG for any belt component that contains an abandoned bag.

If the transaction ends and there are no abandoned bags, but the application left the SBD in the enabled state, then the platform will disable the insertion point on the device (closing physical barrier if present.)

Local airport requirements may impose specific requirements on the behaviour of the CUSS platform and Common Launch Application when abandoned bags are detected. It is a CUSS platform implementation requirement to be aware of and comply with any such requirement, including setting the kiosk out of service and notifying airport staff.

It is possible that an SBD and/or airport belt system has a mechanism for forwarding and diverting abandoned bags away from self bag drop positions automatically. It is acceptable for a CUSS platform and SBD to make use of any such capability or mechanism as long as this process does not require additional business logic within the tenant CUSS applications.

Once the abandoned bags have been removed or diverted (by manual intervention or automated mechanism) the CUSS platform shall report the appropriate events (BAGGAGE_ABSENT, etc.)

If there are any local requirements that a Self Bag Drop position be set out of service, that local airport staff be notified, or that any other specific action be taken in response to an abandoned bag, it is the platform provider's responsibility to implement those requirements.

The CUSS Technical Specification *recommends* that applications deployed in shared self bag drop systems remain UNAVAILABLE in case of abandoned bags. However, it is a CUSS application business logic decision whether to remain in the AVAILABLE state when abandoned/unexpected bags are reported by the platform and, if activated in this state, how to carry out a transaction or recover the abandoned bags.

7.17.5 Receipt and Heavy Tag Printing

Please refer to section 6.4 for information on how specialty document printing, such as baggage receipts and heavy tag printing, should be accomplished using the existing CUSS General Purpose Printer (GPP) capability for SVG and PDF documents.

These specialty GPP printers will be their own component group and not linked to the SBD components.

In particular, if a CUSS kiosk supports heavy tag printing or baggage receipt printing as part of its self bag drop device, the CUSS platform shall:

4. Include a GPP printer definition as set in Section 7.11 for each specialty printer
5. Ensure that the components' Characteristics about paper size and orientation are accurate for each printer
6. Include the characteristics keyword DS_TYPES_HEAVYTAG as part of a heavy tag printer component's Manufacturer.firmwareVersion setting.

As well, a kiosk provider shall publish information about the formatting and size requirements for their heavy tag printer to all airlines deploying on the SBD kiosk. This is important because, **at the time of publication of this CUSS Technical Specification there is no industry standard for specialty/heavy tag printing.**

Please refer to Appendix G for more information.

7.17.6 Standard Operations, Behaviour, and Sequence Diagrams

This section attempts to document how the CUSS Technical Solution Groups expects CUSS platform and application implementations to carry out common transaction tasks with self bag drop devices.

By providing examples, it is intended to answer many questions about the interpretation of the CUSS-SBD specification and how to use it in practice, and by providing key aspects to be aware of for each case.

Detect of the SBD and attached airport baggage system are operating correctly

- As with all CUSS devices, it is an application business logic decision to monitor the condition of the SBD device and set the application UNAVAILABLE or AVAILABLE in response to various device conditions.

- Different SBD device with have different levels of equipment, capability, and error detection. When a CUSS platform reports an error condition on an SBD, this reflects the inability of the local device to function and does not merely mean, for example, that the airport belt is temporarily congested.

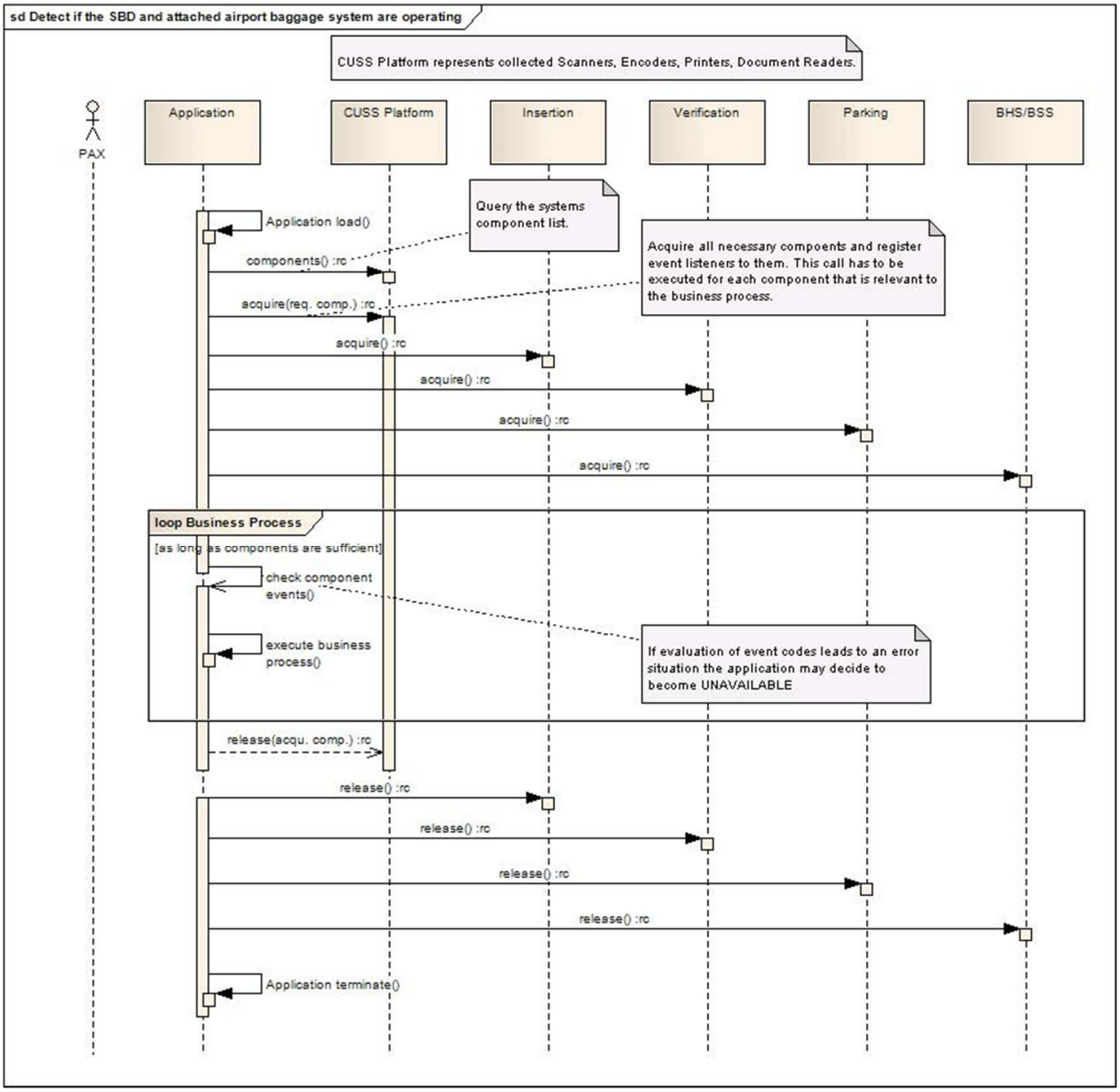
- The CUSS-SBD component group may include an *optional* BHS/BSM. If the platform includes this component, that component's status will indicate the condition of the airport belt system directly. If the platform does not include this component, the remaining SBD components will change to an error state if there is an outage in the airport system that prevents the SBD from functioning.

- Airport baggage systems are quite complex, with numerous levels of monitoring at the PLC level and via SCADA systems. CUSS applications running on self bag drop kiosks are not directly involved in this monitoring, and should instead focus on the errors reported directly by the CUSS platform.

- Many providers of bag processing equipment provide a granular state that is a de facto standard for baggage equipment monitoring. For reference, here is how those states are represented by CUSS, if they occur on equipment that directly influences the CUSS-SBD component(s):

Priority	General State	CUSS Status
1	Safety Stop	BAGGAGE_EMERGENCY_STOP
2	Unknown/Comms Fault	NOT_RESPONDING
3	Offline/Manual/Out of Service	NOT_RESPONDING
4	Error	HARDWARE_ERROR
5	Warning	OK
6	Die-back	OK
7	Stopped in Auto	OK
8	Stopped	OK
9	Starting	OK
10	Energy Save	OK
11	Redundancy	OK

12	Full	BAGGAGE_FULL
13	Started/Running/Enabled	OK
14	Power Up	HARDWARE_ERROR



Determine if there is room on the belts to start or continue processing

- Even if the SBD is operating without technical error, applications must still review the condition of each belt processing component to at all times be aware of where bags are and if processing is available.

- The public bag capacity events reported for each belt component can be used to monitor the status of the belts. In particular, the events BAGGAGE_ABSENT, BAGGAGE_PRESENT, BAGGAGE_FULL, and BAGGAGE_UNEXPECTED_BAG are critical to determining room on the belt.

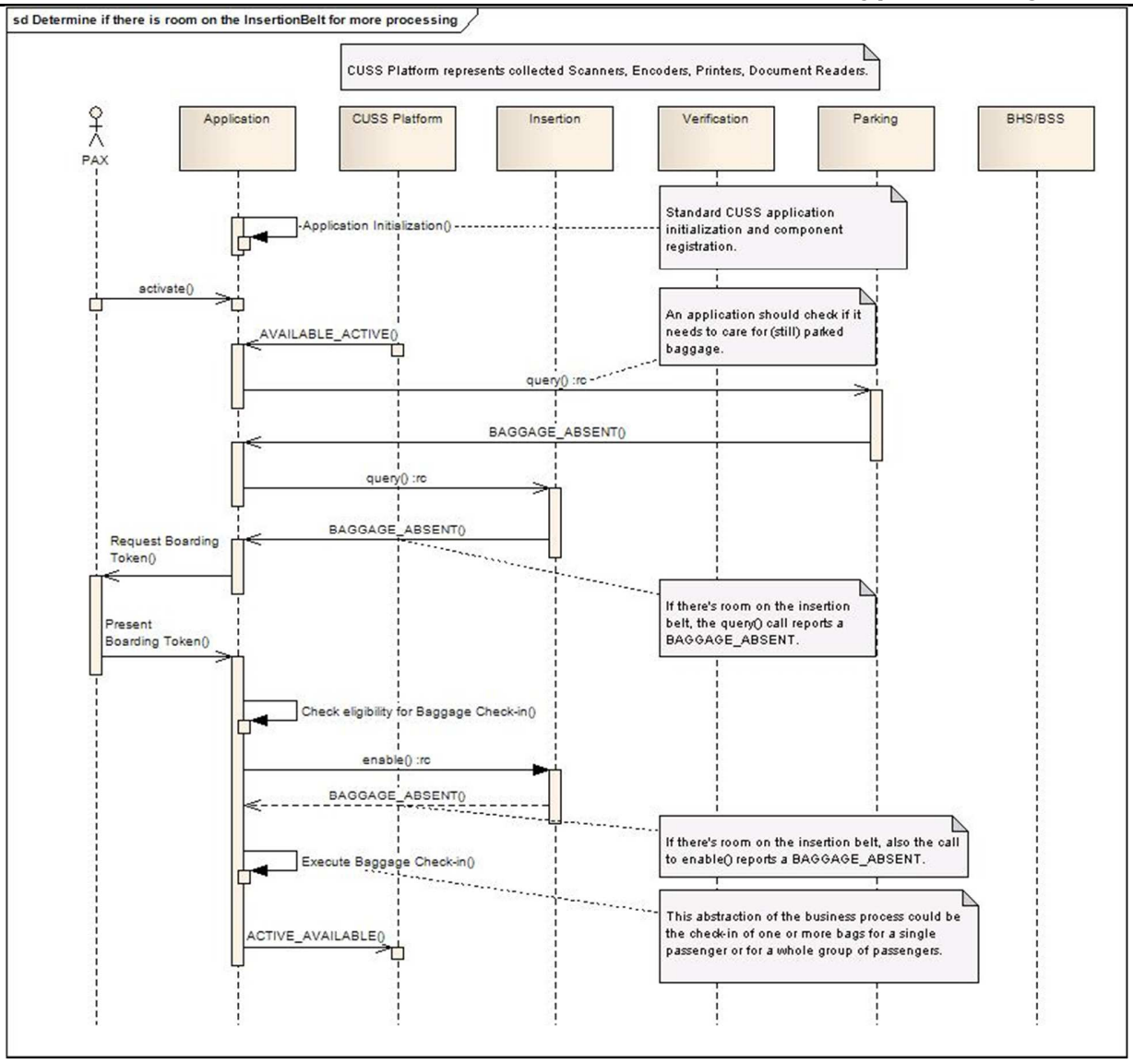
- Applications should be aware that some Self bag Drop devices support multiple bags simultaneously, and in writing applications to fully support this capability careful monitoring of all status codes (BAGGAGE_PRESENT vs. BAGGAGE_FULL, for example) is important.

- Depending on whether an application is idle, or how far along in the transaction the application is, there may be different business logic in effect. For example, in some cases it makes sense to wait up to 60-90 seconds for a belt to become free, in others it might make sense to cancel a transaction already in progress.

- Be aware that when the kiosk is idle or even at the start of a new airline application transaction, there may still be bag(s) on the parking belt from a previous transaction. These bags are under airport belt control and do not need to be processed by the new transaction, so there is no need to delay the transaction until the parking belt is completely free.

- Always check for bag placement and belt status prior to doing forward() and backward() requests.

- Avoid ending a transaction and leaving unprocessed bags on the belt. Always try and direct passengers to recover their bags and wait for this to happen, even if transactions are canceled. Bags abandoned at the end of the transaction will likely put the kiosk out of service and require supervisor intervention.



Enable the bag drop devices and monitor for user interference

- Different bag drop devices will have different capabilities. Review the barrierCapable, backwardCapable and similar Characteristics to determine the type of device in use.

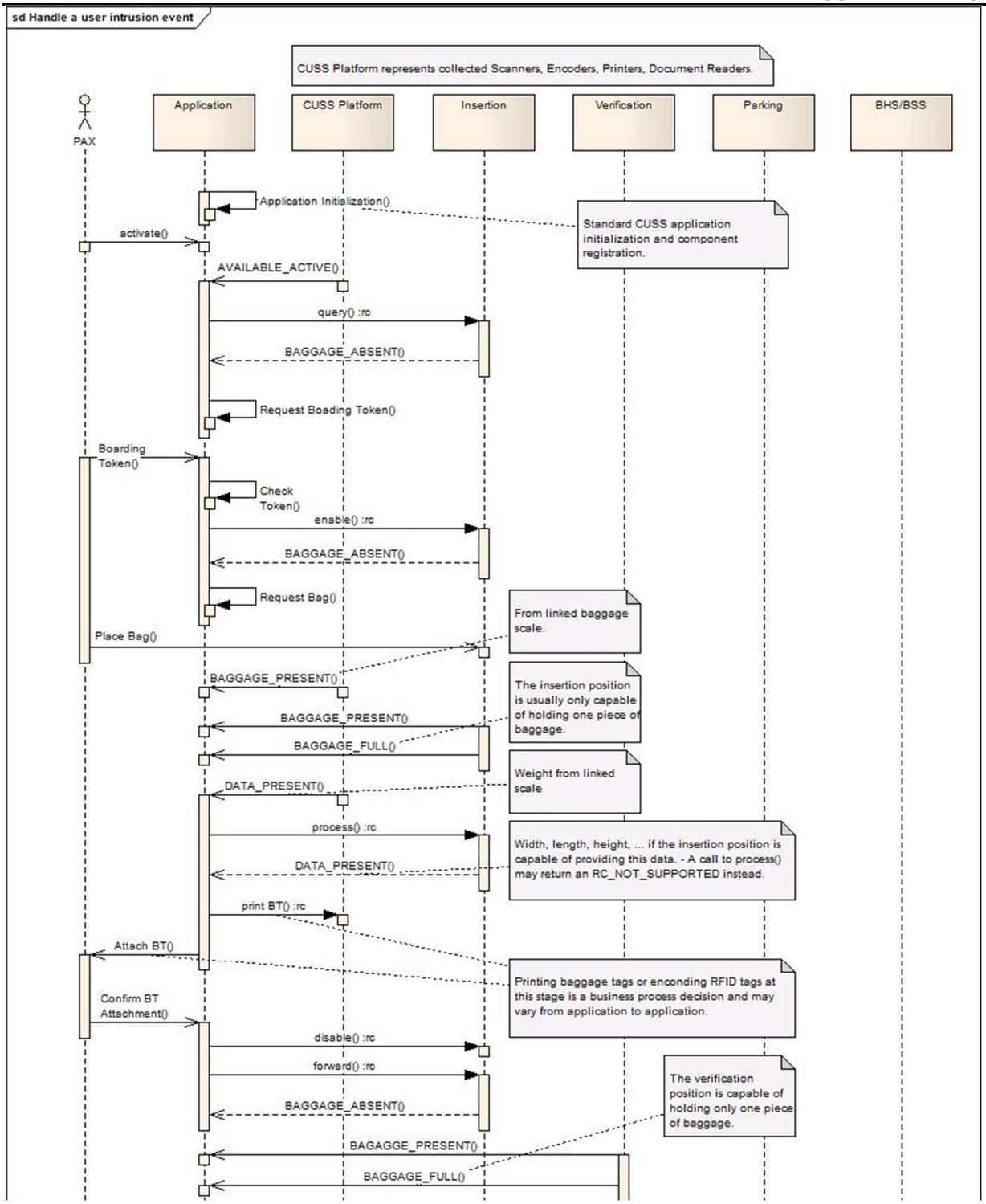
- To allow passengers to deposit bags on the device, always call the enable() directive as this is required to activate a safety barrier, disable interference sensors, or other tasks required to prepare the physical device. Once the bag is accepted, call disable() to prevent adding a second bag or interference with the verification process.

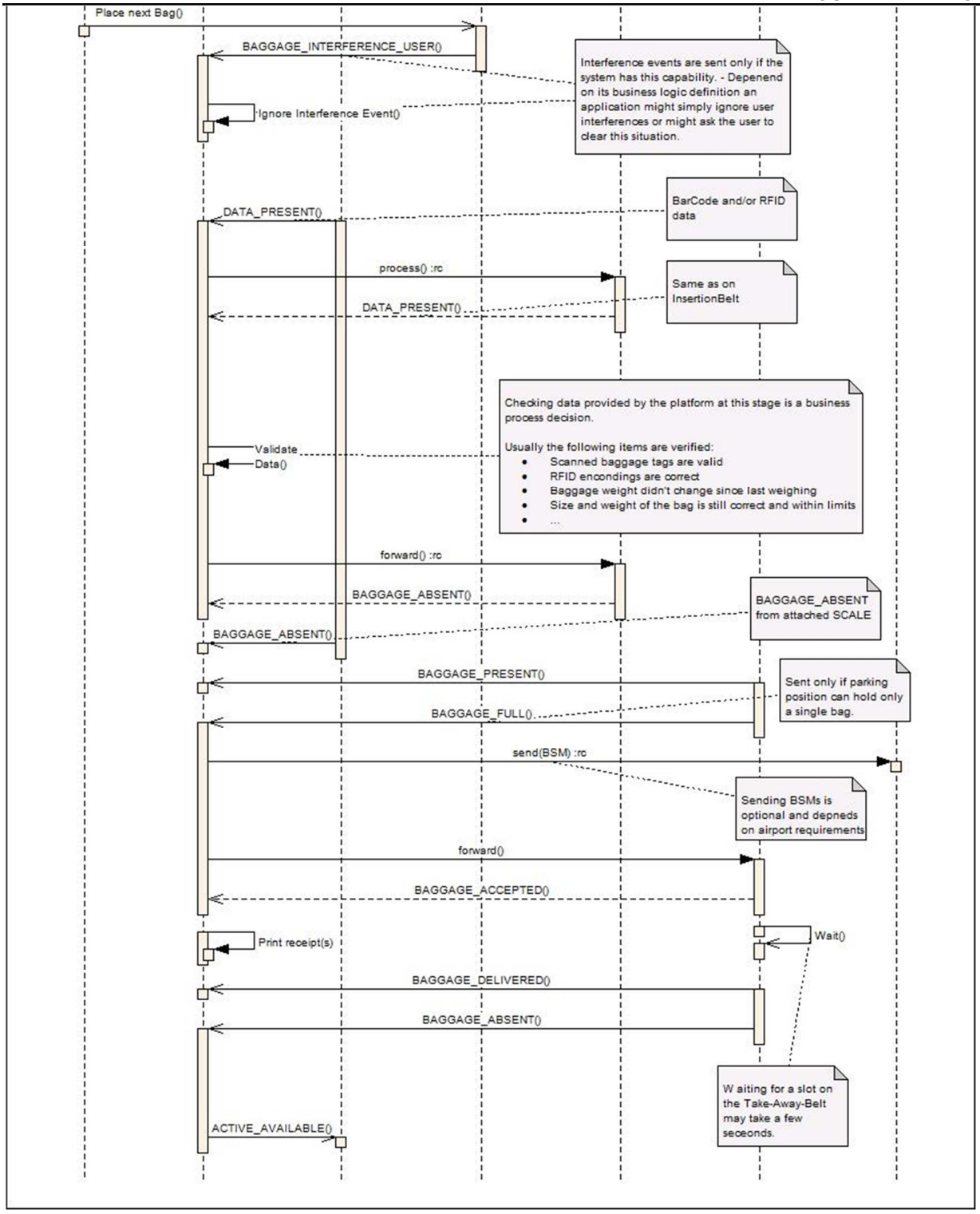
- For similar reasons, when returning a bag back to the passenger (for a cancelled transaction or similar), always use the offer() directive.

- While processing bags, monitor the SBD components for BAGGAGE_INTERFERENCE_USER, BAGGAGE_UNEXPECTED_CHANGE, and similar events that could indicate the passenger is trying to manipulate the bag.

- The application should decide if it wants to cancel/restart process of the bag when user interference is detected. Note that because user interference events are not related to safety and security, the platform may not necessarily stop bag movement in response to these events.

- Usually there is no need to cancel a transaction when user interference is detected. Waiting for a certain period of time for the passenger to cease the interference may be sufficient. However overall, how to handle these events is always an application business logic decision.


sd Handle a user intrusion event
CUSS Platform represents collected Scanners, Encoders, Printers, Document Readers.
PAX
Application
CUSS Platform
Insertion
Verification
Parking
BHS/BSS
Application Initialization()
Standard CUSS application initialization and component registration.
activate()
AVAILABLE_ACTIVE()
query() :rc
BAGGAGE_ABSENT()
Request Boading Token()
Boarding Token()
Check Token()
enable() :rc
BAGGAGE_ABSENT()
Request Bag()
From linked baggage scale.
Place Bag()
BAGGAGE_PRESENT()
The insertion position is usually only capable of holding one piece of baggage.
BAGGAGE_PRESENT()
BAGGAGE_FULL()
Weight from linked scale
DATA_PRESENT()
process() :rc
Width, length, height, ... if the insertion position is capable of providing this data. - A call to process() may return an RC_NOT_SUPPORTED instead.
DATA_PRESENT()
print BT() :rc
Attach BT()
Printing baggage tags or enconding RFID tags at this stage is a business process decision and may vary from application to application.
Confirm BT Attachment()
disable() :rc
forward() :rc
The verification position is capable of holding only one piece of baggage.
BAGGAGE_ABSENT()
BAGAGGE_PRESENT()
BAGGAGE_FULL()


Place next Bag()
BAGGAGE_INTERFERENCE_USER()
Interference events are sent only if the system has this capability. - Depenend on its business logic definition an application might simply ignore user interferences or might ask the user to clear this situation.
Ignore Interference Event()
BarCode and/or RFID data
DATA_PRESENT()
process() :rc
Same as on InsertionBelt
DATA_PRESENT()
Checking data provided by the platform at this stage is a business process decision.
Usually the following items are verified:
Scanned baggage tags are valid
RFID encondings are correct
Baggage weight didn't change since last weighing
Size and weight of the bag is still correct and within limits
...
Validate Data()
forward() :rc
BAGGAGE_ABSENT()
BAGGAGE_ABSENT from attached SCALE
BAGGAGE_ABSENT()
BAGGAGE_PRESENT()
Sent only if parking position can hold only a single bag.
BAGGAGE_FULL()
send(BSM) :rc
Sending BSMs is optional and depneds on airport requirements
forward()
BAGGAGE_ACCEPTED()
Wait()
Print receipt(s)
BAGGAGE_DELIVERED()
BAGGAGE_ABSENT()
W aiting for a slot on the Take-Away-Belt may take a few seceonds.
ACTIVE_AVAILABLE()

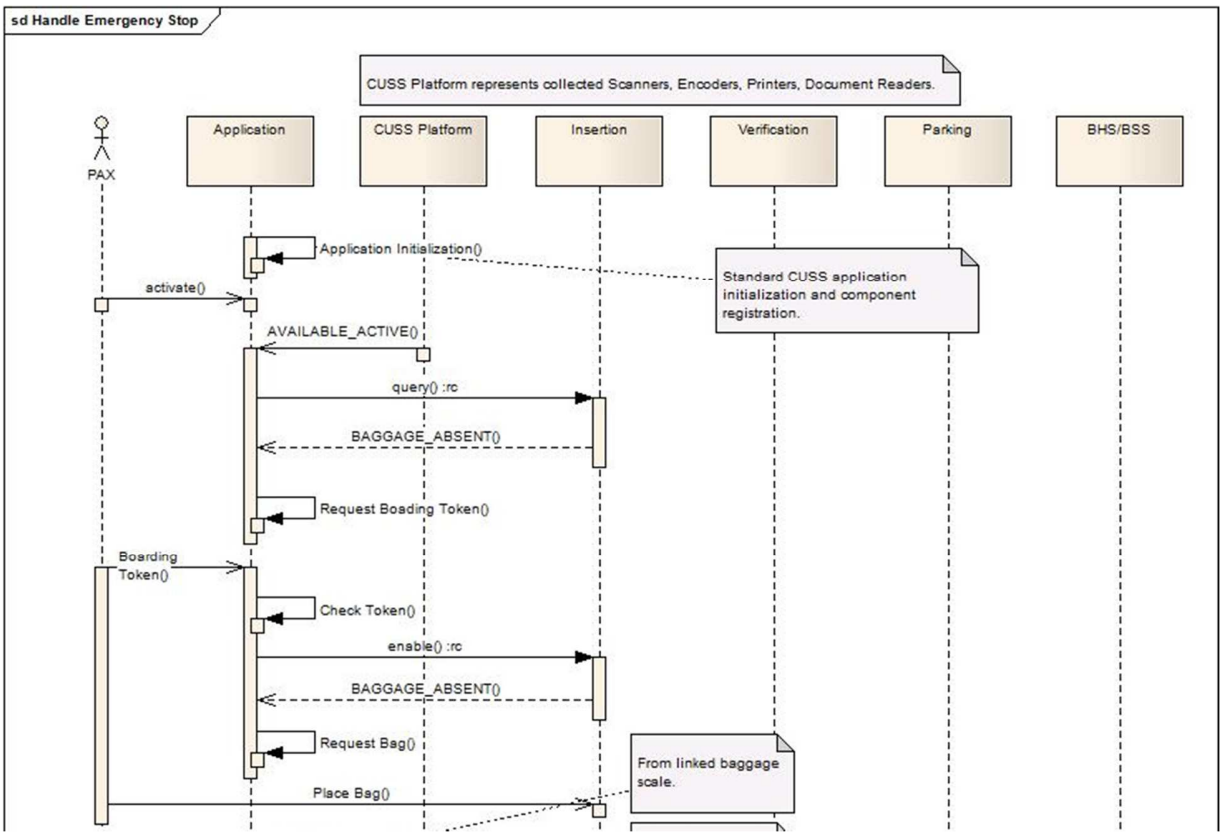
React to critical conditions like safety intrusions, emergency stops, and mistracked bags

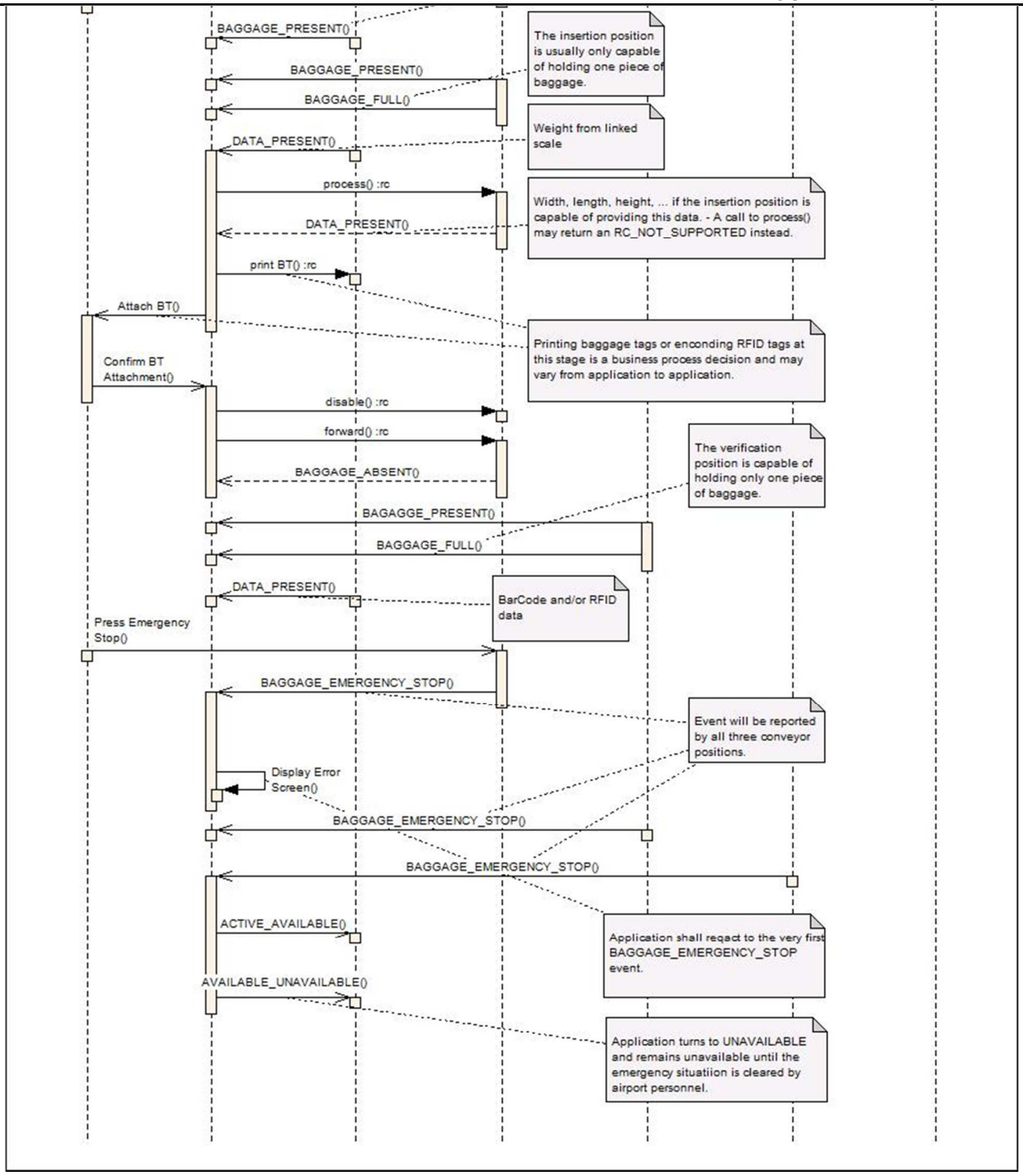
- While processing bags, monitor the SBD components for BAGGAGE_INTRUSTION_SAFETY, BAGGAGE_EMERGENCY_STOP, BAGGAGE_RESTLESS, BAGGAGE_MISTRACKED, and similar critical error conditions. These events could indicate someone or something is in danger.

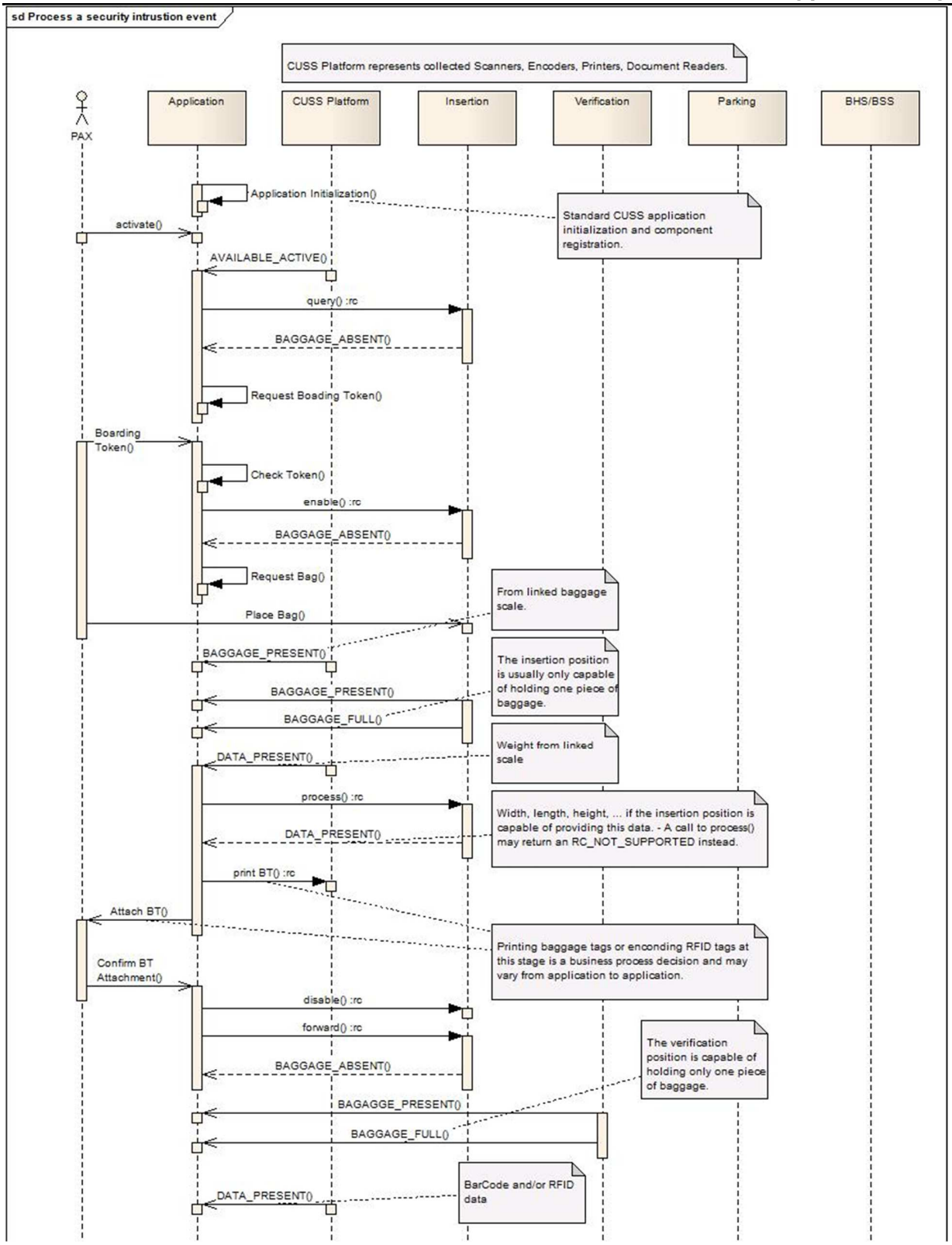
- Note that because these intrusion/security events are related to the ongoing safety and security of the passenger and of the airport, the platform will automatically stop any bag or belt movement in response to these events. There is no need for the application to do so (though there is no harm in making the request.)

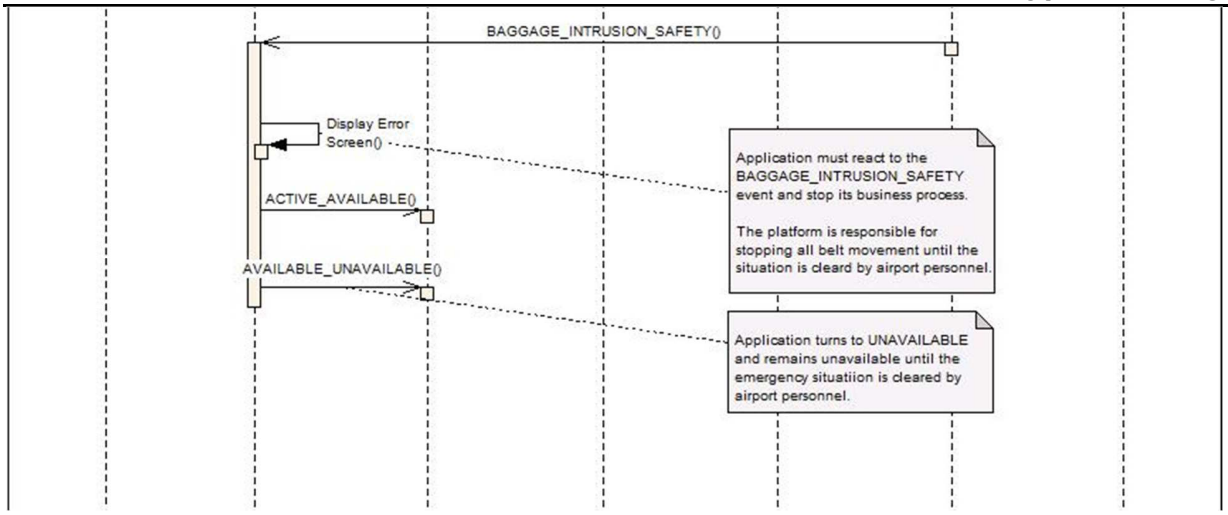
- Usually these conditions will only be resolved when a supervisor completes manual inspection of the SBD and re-actives it, which may take a few minutes. Until that takes places the SBD device may remain completely unavailable. For example, an emergency stop button must halt all belt movement and no belts can move until the situation has been reviewed and the system recovered.

- In most cases, the application will need to abandon the transaction in progress when these events occur, then go to UNAVAILABLE, and monitor the SBD components and return to the available state when the situation has recovered.




BAGGAGE_PRESENT()
The insertion position is usually only capable of holding one piece of baggage.
BAGGAGE_PRESENT()
BAGGAGE_FULL()
Weight from linked scale
DATA_PRESENT()
process() :rc
Width, length, height, ... if the insertion position is capable of providing this data. - A call to process() may return an RC_NOT_SUPPORTED instead.
DATA_PRESENT()
print BT() :rc
Attach BT()
Printing baggage tags or enconding RFID tags at this stage is a business process decision and may vary from application to application.
Confirm BT Attachment()
disable() :rc
forward() :rc
The verification position is capable of holding only one piece of baggage.
BAGGAGE_ABSENT()
BAGAGGE_PRESENT()
BAGGAGE_FULL()
DATA_PRESENT()
BarCode and/or RFID data
Press Emergency Stop()
BAGGAGE_EMERGENCY_STOP()
Event will be reported by all three conveyor positions.
Display Error Screen()
BAGGAGE_EMERGENCY_STOP()
BAGGAGE_EMERGENCY_STOP()
ACTIVE_AVAILABLE()
Application shall reqact to the very first BAGGAGE_EMERGENCY_STOP event.
AVAILABLE_UNAVAILABLE()
Application turns to UNAVAILABLE and remains unavailable until the emergency situatiion is cleared by airport personnel.


sd Process a security intrustion event
CUSS Platform represents collected Scanners, Encoders, Printers, Document Readers.
PAX
Application
CUSS Platform
Insertion
Verification
Parking
BHS/BSS
Application Initialization()
Standard CUSS application initialization and component registration.
activate()
AVAILABLE_ACTIVE()
query() :rc
BAGGAGE_ABSENT()
Request Boading Token()
Boarding Token()
Check Token()
enable() :rc
BAGGAGE_ABSENT()
Request Bag()
From linked baggage scale.
Place Bag()
BAGGAGE_PRESENT()
The insertion position is usually only capable of holding one piece of baggage.
BAGGAGE_PRESENT()
BAGGAGE_FULL()
Weight from linked scale
DATA_PRESENT()
process() :rc
Width, length, height, ... if the insertion position is capable of providing this data. - A call to process() may return an RC_NOT_SUPPORTED instead.
DATA_PRESENT()
print BT() :rc
Attach BT()
Printing baggage tags or enconding RFID tags at this stage is a business process decision and may vary from application to application.
Confirm BT Attachment()
disable() :rc
forward() :rc
The verification position is capable of holding only one piece of baggage.
BAGGAGE_ABSENT()
BAGAGGE_PRESENT()
BAGGAGE_FULL()
BarCode and/or RFID data
DATA_PRESENT()



Keep track of when bags move from the SBD to the airport belt

- In order to move a bag onto the airport belt, the CUSS application will need to move a bag forward from the insertion belt, to the verification belt, onto the parking belt, and finally to the airport belt.

- Even though each belt might support multiple bags, bags will procede through the SBD in sequential (FIFO) fashion.

- While not all SBD devices will have three separate physical belts (many will only have one or two), the CUSS-SBD interface will always include all three logical belts for processing, and the CUSS application must interact with all three belt components.

- The CUSS platform will ensure that even if the kiosk is physically equipped only with a one or two-belt self bag drop device, the device will logically be implemented with all three belts defined in this technical specification.

- Generally speaking, CUSS applications should not require specific logic depending on whether the physical SBD has one, two, or three physical belts, so long as the application is written to comply with this CUSS-SBD standard.

- Make note of the Characteristics of each belt component. Some systems only allow forward bag movement. Some systems allow for more than one bag per belt, particularly for the parking belt. It's up to the application business logic to detect and adapt to these situations to provide the best possible transaction capability to its customers.

- Once a bag is on the parking belt it is usally ready to be promoted to the airport belt. Before asking to move the bag on, however, the application will likely need to create or update a BSM message telling the airport about the bag. If the optional BHS CUSS-SBD component that supports

DS_TYPES_RP1745 is present then the CUSS application can send a BSM directly with this component. Any other appropriate or existing method to issue/update the BSM is also acceptable.

- To promote the bag, use the forward() directive with the appropriate timeout to move the bag onto the airport belt. The response will either be BAGGAGE_DELIVERED if the bag is physically on the airport belt, or BAGGAGE_ACCEPTED if it remains on the parking belt but is under the control of the airport belt system (for example, waiting for a logical induction slot.)

- Once a bag is accepted, an application can continue processing more bags, or end the transaction and return to the available state even if the bag is still on the parking belt. Once the parking belt is empty, the application will receive a BAGGAGE_ABSENT event.

Check if bags have changed between the insertion and verification points

- The application can use the process() directive to tell the SBD to activate belt mechanisms to read and send information about the bag to the application.

- Depending on the SBD, this process() directive will trigger different data for different belts. For example, in some cases the scale device or barcode scanner might be activated on both the insertion belt and the verification belt, and in others they may only be available on the verification belt.

- Applications should not be written to assume one way or another, and should look for RC_NOT_SUPPORTED as a response to the process() directive on each belt.

- The platform will return data that can be read from the bag using distinct data components. Each one will trigger a DATA_PRESENT (or DATA_MISSING) in response to the process() directive, as soon as information is available:
  - Weigh and Alibi number → scale component
  - Dimensions → belt component
  - Barcodes → barcode scanner component
  - RFID tags → RFID reader component

- It is the application's business logic and responsibility to compare values and detect if anything changed. The platform will not decide this on behalf of the application.

- Depending on capabilities, some SBD devices may be able to detect changes using other methods, such as advanced volumetrics or photoanalysis. In this case the application can process the BAGGAGE_UNEXPECTED_CHANGE event.

- Depending on what change is detected, the application can decide to move the bag back (if the belt is backwardCapable) and prompt the user to restart the process. Once a bag has successfully passed verification, it can finally be moved to the parking belt. All these are application business logic choices.

- The CUSS platform will ensure that even if the kiosk is physically equipped only with a one or two-belt self bag drop device, the device will logically be implemented with all three belts defined in this technical specification.

- Generally speaking, CUSS applications should not require specific logic depending on whether the physical SBD has one, two, or three physical belts, so long as the application is written to comply with this CUSS-SBD standard.

- Make note of the Characteristics of each belt component. Some systems only allow forward bag movement. Some systems allow for more than one bag per belt, particularly for the parking belt. It's up to the application business logic to detect and adapt to these situations to provide the best possible transaction capability to its customers.

Handling a mistracked or unexpected bag condition, or an error promoting a bag to the airport belt

- Mistracked bags and unexpected bags are usually critical errors, and the application will not be able to complete the transaction. They should be processed similar to a critical error like a restless bag or an emergency stop.

An example of end-to-end process of a single bag successfully without abnormal conditions

- It is anticipated that a main priority of airline self bag drop applications is to ensure that a successful bag drop can take place quickly and without error, in a fashion that is intuitive to the passenger.

- Aside from the conditions above, what is considered a critical error, temporary error, or insignificant condition is the business logic choice of an application. For example, and application that does not charge by bag weight might not care if the weight changed slightly at the verification point.

- It is an application business logic decision whether or not to (re)print tags at the self bag drop, whether to print heavy tags, the maximum number of bags to process for a customer, to calculate allowances by weight, piece, or pool concepts, and any similar tasks and responsibilities of airline baggage acceptance.

- The CUSS platform and SBD will not dictate or impose any of these rules, except for aspects required for local integration, such as Health & Stafety weight and size limits.

- For a sample sequence diagram, see the "Typical sequence" example at the start of this section.

The passenger decides to cancel the transaction mid process

- A customer might abandon a transaction mid way through (walk away or be distracted), or might choose to explicitly cancel a transaction for various reasons (unanticipated bag fees, repack, etc.)

- A transaction might need to be cancelled as a result of a business rule as well, such as a failed payment transaction, weight or size issues, or a change in the flight status.

- **It is an application business logic requirement to properly "unwind" the transaction as the result of a transaction cancellation.** This includes moving bags back to the insertion point (if supported), offering the bag back to the customer, using visual and audia prompts to get the attention of the user, and similar.

- It is very important that applications try to avoid any case the results in an abandoned bag, as described in a previous section, as this may cause the self bag drop device to be out of service until a supervisor is able to recover the abandoned bags.

7.18 Independent Baggage Scale

Description of Device:

For CUSS kiosks providing baggage tag printing only, the BaggageScale interface can be used to weigh a bag for registering baggage with the DCS system (and to encode weight information on the baggage tag if necessary).

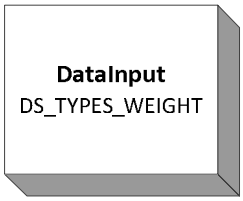
A BaggageScale does not convey baggage in any form. For baggage scales included in Integrated Baggage Conveyors, see Section 7.16. A kiosk may be connected to a Baggage Scale and an Integrated Baggage Conveyor at the same time.

A separatate component definition is needed for simple baggage scales, as the Self Bag Drop extension for AEA used by the Integrated Conveyor System components, does not support scale-only devices as of AEA2012-2.

Important Note: This component definition replaces the **Conveyor** component definition included in CUSS-TS 1.2. As of CUSS 1.3, the previous **Conveyor** interface is deprecated (targeted for removal in a future release) even though it remains in the IDL files.

Virtual Component Linking Diagram:

Dedicated Baggage Scale (non-Conveyor)



Distinct Characteristics:

BaggageScale	
Characteristic	**Value**
Manufacturer.FirmwareVersion	This will include the indicator DS_TYPES_WEIGHT to confirm the device represents a dedicated scale. This indicates weight information will be conveyed in accordance with the CUSS.SBD.XSD messaging schema.

Distinct Status Conditions:

Code	Meaning
BAGGAGE_ABSENT	No item is on the scale and the scale is stable at its zero.
BAGGAGE_PRESENT	There is an item on the scale and a stable weight has not yet been read
BAGGAGE_WEIGHT_OUT_OF_RANGE	An item is on the scale but its weight is above or below the limit that can be read by the scale.

DATA_PRESENT	There is an item on the scale and a stable weight (and alibi number, if applicable) is available.

7.18.1 Device Component Interface Directives Extensions

This chapter defines the relation between new status codes and the baggage scale virtual component directives. The relations of the standard CUSS status codes remain as they are defined in chapter [*Device Component Interface (DCI) Directives*].

Virtual Component Type: **BaggageScale** Status Code	**acquire()**	**cancel()**	**disable()**	**query()**	**test()**	**setup()**	**release()**
BAGGAGE_ABSENT			X	X	X		
BAGGAGE_PRESENT			X	X	X		
BAGGAGE_WEIGHT_OUT_OF_RANGE			X	X	X		

The CUSS platform shall respond BAGGAGE_ABSENT instead of OK to query() requests when no bag is present and no other error condition is detected.

Notwithstanding the above requirement for CUSS platforms, for compatibility reasons it is recommended that CUSS applications should accept and interpret OK and BAGGAGE_ABSENT as equivalent.

BAGGAGE_ABSENT and BAGGAGE_PRESENT are _public events.

7.18.2 Data Format (DS_TYPES_WEIGHT)

The BaggageScale component derives from the DataInput component, and delivers the weight information as part of the BAGGAGE_RESTLESS, BAGGAGE_PRESENT and BAGGAGE_ABSENT events, using a CUSS msgDataType with a single dataRecord.

The DataInput component for the scale delivers its data in a CUSS msgDataType with a single dataRecord in it. Regardless of the reporting units or precision of the scale component, the platform must convert (if needed) and report the weight in metric grams. A weight of 0 grams is a valid weight.

Format specification and example data: **msgDataType.records[0] XML message [CUSS.SBD.XSD]**

```
<?xml version="1.0" encoding="UTF-8"?>
<baggageData xmlns="urn:CUSS-1.3/types/conveyorInterface"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <dimension>
            <load>
                  <weight>18000</weight>
                  <alibi>
                        <value>0123456789-ABCDEF</value>
                  </alibi>
```

```
                        <stable>true</stable>
                    </load>
                </dimension>
            </baggageData>
```

The component may report a 0 or non-zero weight in a BAGGAGE_ABSENT condition, depending on local regulations and calibration of the scale. For example, some jurisdictions may require that weights under a certain threshold be omitted.

As part of commercial Weights & Measures regulations, some jurisdiction require that scales used for commercial transactions also provide a measurement tracking reconciliation number, sometimes called an alibi number, along with the measurement value.

If the scale and location require the use of alibi numbers, it is a platform requirement to provide the alibi reference to the application. If provided, the alibi number must be included in the appropriate field of the XML message reported to the application, as defined in CUSS.SBD.XSD.

It is a platform implementation choice and task to determine how to generate, obtain, and track/audit weights and alibi numbers, and all other aspects of compliance with local or Weights & Measures requirements relating to alibi numbers.

It is an application business logic decision to detect and properly use alibi numbers provided by the CUSS platform. Usage requirements and alibi number syntax may vary from location to location, and cannot be described by the CUSS Technical Standard.

If a scale is able to detect unstable weight conditions, this condition shall be reported to the application as a BAGGAGE_PRESENT event, which will *not* include any weight value. In this case, as the weight changes over time, the expected sequence of events is:

BAGGAGE_ABSENT	(no bag)
BAGGAGE_PRESENT	(unstable weight)
DATA_PRESENT	(stable bag)
BAGGAGE_PRESENT	(bag unstable during adjustment or item added/removed)
DATA_PRESENT	(stable bag)
BAGGAGE_PRESENT	(bag unstable during removal)
BAGGAGE_ABSENT	(no bag)

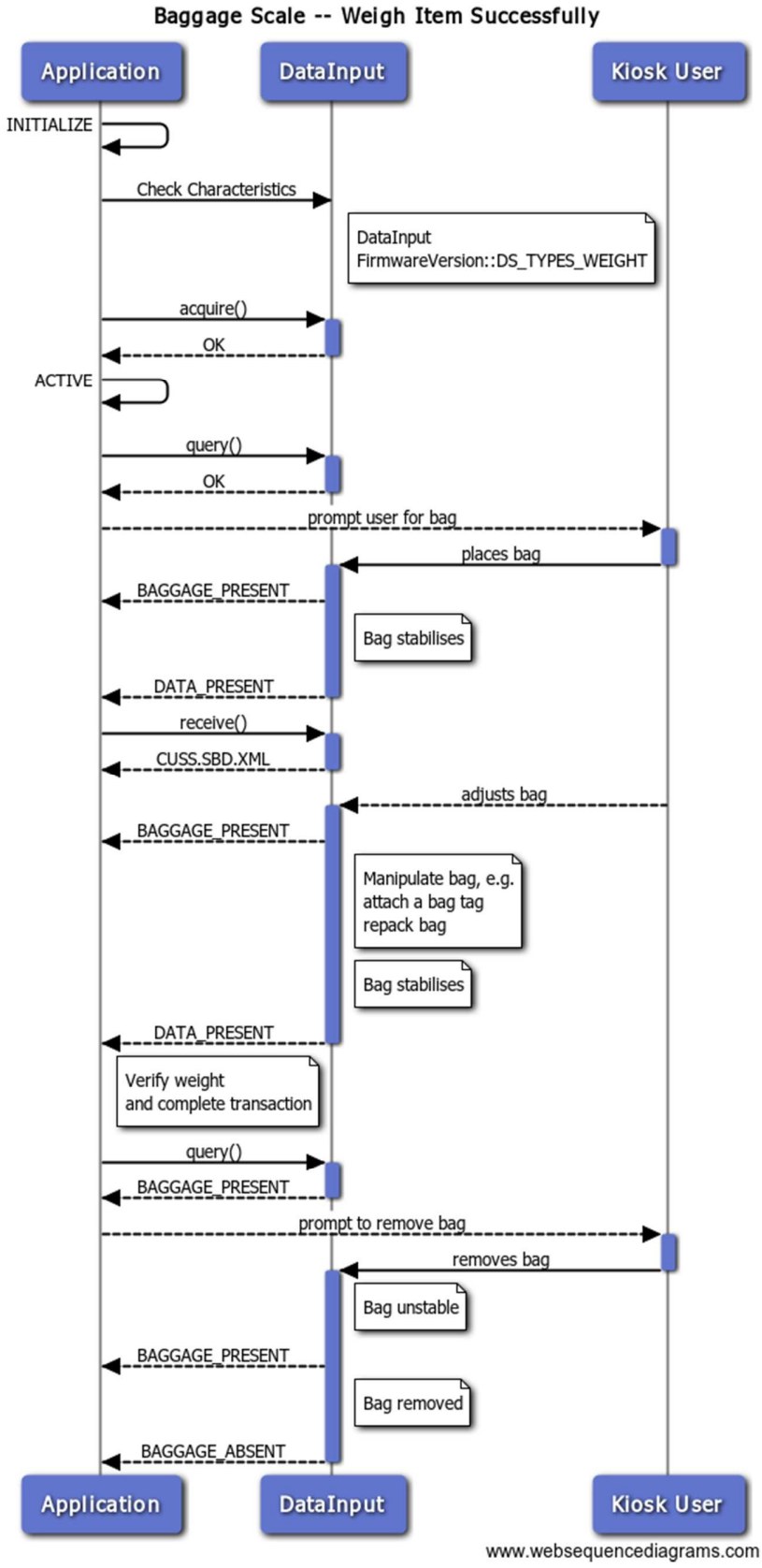
The CUSS technical specification does not require that scale be able to report unstable weights. A scale that cannot detect unstable weight conditions but always reports a live weight, may not broadcast the the BAGGAGE_PRESENT condition and would instead immediately report DATA_PRESENT.

BAGGAGE_ABSENT	(no bag)
DATA_PRESENT	(live weight of unstable bag)
DATA_PRESENT	(stable bag)
DATA_PRESENT	(live weight of unstable bag during adjustment or item added/removed)
DATA_PRESENT	(stable bag)
DATA_PRESENT	(live weight of unstable bag during removal)
BAGGAGE_ABSENT	(no bag)

Some CUSS platforms may choose to simulate the detection of unstable weight by providing BAGGAGE_PRESENT immediately prior to the DATA_PRESENT in the above scenario.

Applications using the scale interface should accommodate either situation as part of the scale transaction business logic. Applications should also be able to accommodate situations where the scale reports many weight values in a row during instability while the weight is changing.

7.18.3 Typical Sequence Diagram



7.19 Generic Payment Device

Description of Device:

The Payment Device Interface is added to CUSS 1.3 as an optional kiosk component. As the industry moves away from payment transactions based on magnetic cards, these transactions are now designed to be carried out via a “Generic Payment Device”.

CUSS 1.3 kiosks and this technical specification still provide a magnetic card reader interface that can be used directly for payment transactions and for FOID transactions.

The goal of a generic payment device is that the kiosk contains a “black box” payment device (such as a Chip & PIN terminal, for example) that carries out a payment transaction basic on input from the application (such as the amount of the transaction) without exposing sensitive payment data to the CUSS platform or application.

The Generic Payment Interface allows these types of transactions:

1. Authorization with automatic commit
2. Authorization with manual commit
3. Authorization with manual cancel
4. Pre-authorization with final amount post-authorization
5. Pre-authorization with manual cancel

The Generic Payment Interface has these primary design goals:

1. Allow generic payment transactions such as EMV/chip & PIN in the modes listed above
2. Define an interface that also allows non-payment FOID magnetic cards to be read
3. Complete payment without exposing the application to any sensitive payment information
4. Allow the application to provide extended billing information for tracking and reconciliation
5. Use a flexible XML messaging format for exchanging payment transaction information
6. Include the flexibility needed to be deployed to kiosks with an existing proprietary interface
7. Defer receipt printing requirements to the application to aid in compliance and certification
8. The interface supports shared systems payment models that are to be deployed to CUSS systems (including payment aggregators, multi-merchant multi-acquirer solutions, and any other viable technical solution.)

Important Notices:

The CUSS Technical Specification defines only the *interface* between the application and the CUSS platform. It does not describe or recommend how the platform should select and integrate components of a Payment Solution Provider to implement the interface.

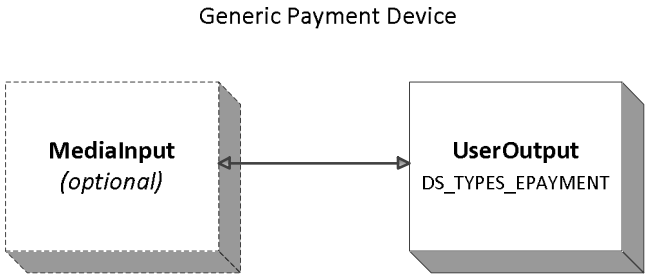
The payment interface allows payment transactions to complete without any sensitive payment information being provided to the application.

The intent of the interface is that platforms must not, under any circumstance, provide sensitive payment information to the CUSS application using any feature of the payment messaging scheme.

In particular, this standard does not list, recommend, or endorse any particular payment hardware or payment solution components that meet the requirements of the interface, nor does it imply that such components would meet “shared payment solution” requirements off-the-shelf without modification.

Thus it is up to the platform supplier to select, modify (if needed) and integrate the payment solution components as required for their CUSS deployment, and to modify their CUSS platform to provide this Payment Interface to the application in order to use and control the selected payment solution.

Virtual Component Linking Diagram:



Description of Virtual Component Linkage:

The Generic Payment Device consists of a single UserOutput virtual component representing the payment device. It may also have a single MediaInput device representing the magnetic card reader integrated with the generic payment device (which could be used as the non-payment card reader on the kiosk.)

The component can be uniquely identified on a kiosk using the Distinct Characteristics listed below. The RealComponentName will contain “EPayment”.

Important Notice:
The CUSS Technical Specification allows a single component to provide payment capabilities, as well as generic (FOID and non-payment) magnetic card reading capabilities. This interface is defined with the optional MediaInput component listed above, in addition to the UserOutput component used for the payment interface.

However, the CUSS Technical Specification defines only the *interface* between the application and the CUSS platform, and does not list or endorse any particular PSP components that allow this behavior. It does not describe or recommend how the platform should select and integrate components of a Payment Solution Provider to implement the interface.

This it is a platform supplier responsibility to investigate and implement this as needed to correctly expose the defined CUSS payment and card reader interfaces. It is possible that the PSP would need to modify or customize their payment terminal to allow the multi-function behavior.

Distinct Characteristics:

The following component types and characteristics can be used to uniquely identify the component representing the Generic Payment Device, amount the list of all device components on the kiosk.

Characteristic	Value
Component type	UserOutput
Manufacturer.FirmwareVersion	Contains the string reference DS_TYPES_EPAYMENT
Manufacturer.FirmwareVersion	Will also contain a capabilities message formatted in XML in accordance with schema CUSS.PAYMENT.XSD.

7.19.1 Data Formats

The following sections describe the data formats to be received or sent to the components of the generic payment device. The basic data format for all data shall be a string (msgDataType) allowing the transmission of XML formatted data structures.

The Generic Payment Device is defined by a single UserOutput component that supports setup() and send() requests from the application, and which can generate custom asynchronous events. The data payload for these requests and events is the standard CUSS msgDataType with the appropriate number dataRecords in it.

The content of each msgDataType record shall be a properly-format XML message corresponding to the schema below and containing the correct XML data for the type of request and transaction the application needs to carry out. The schema is defined by the file CUSS.PAYMENT.XSD.

7.19.2 Application Responsibilities

A complete self-service payment transaction requires many steps. To be clear on certain responsibilities and capabilities, here is some high level guidance on what CUSS applications need do during a payment transaction:

- As part of the setup() request, applications can ***suggest*** *which card brands to accept*, such as VISA, Mastercard, or American Express. The payment solution may or not be able to honour that request, depending on system capabilities. The component characteristics will indicate which card brands can be selected.

- As part of the setup() request, applications can ***suggest*** *which payment media types to accept*, such as magnetic card, EMV, or contactless. The payment solution may or may not be able to honour that request, depending on system capabilities. The component characteristics will indicate which media types can be selected.

- The application ***cannot request*** *to restrict payment only to Debit or Credit* payment card methods.

- It is the application's responsibility to print any required receipts and/or invoices using the other printers available on the kiosk. Because the CUSS application cannot rely on the presence of a receipt printer within the payment device it ***must print the payment receipt*** using the existing CUSS kiosk printer interfaces, based on the receipt data provided by the CUSS payment interface transaction response. The application is responsible for printing because the application typically:

- Owns the payment process requirements, including generating receipts
- Is aware of any airline-specific receipt requirements
- Makes the decision of whether a transaction can continue if receipt printing fails
- Controls the kiosk screen and provides instructions during the entire transaction
- Is ultimately responsible for all aspects of the payment transaction and the end to end customer experience and post-transaction support, including receipts

- **The application cannot issue transaction reversals or chargebacks** from the kiosk (the platform may automatically perform immediate reversals at time of payment, if the application or payment device encounters an error.)

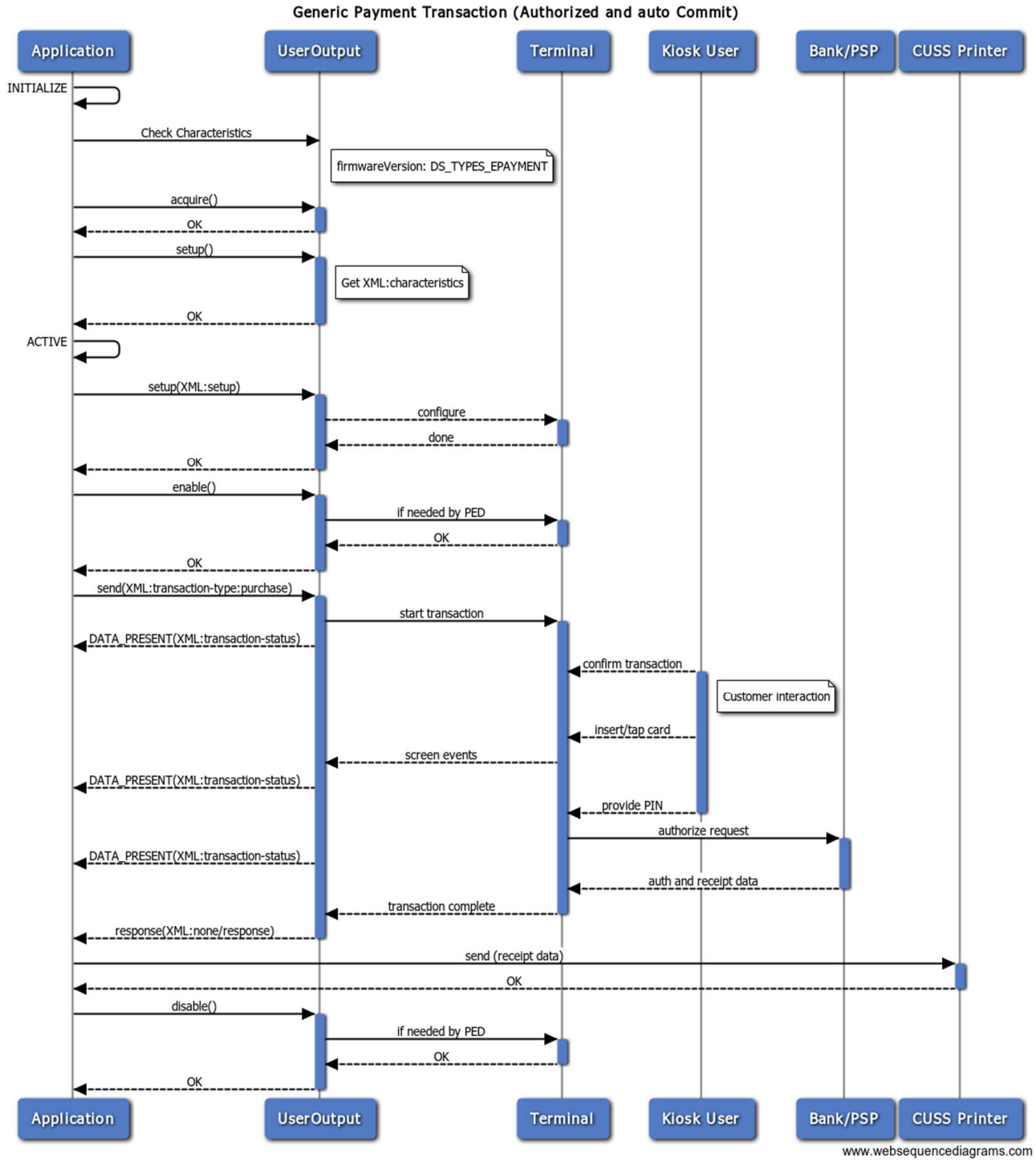
- Because **the CUSS platform must not provide any unmasked or other sensitive payment data** to the application, **the application payment processing cannot expect to retrieve the unmasked payment data** as part of a payment transaction via any method whatsoever.

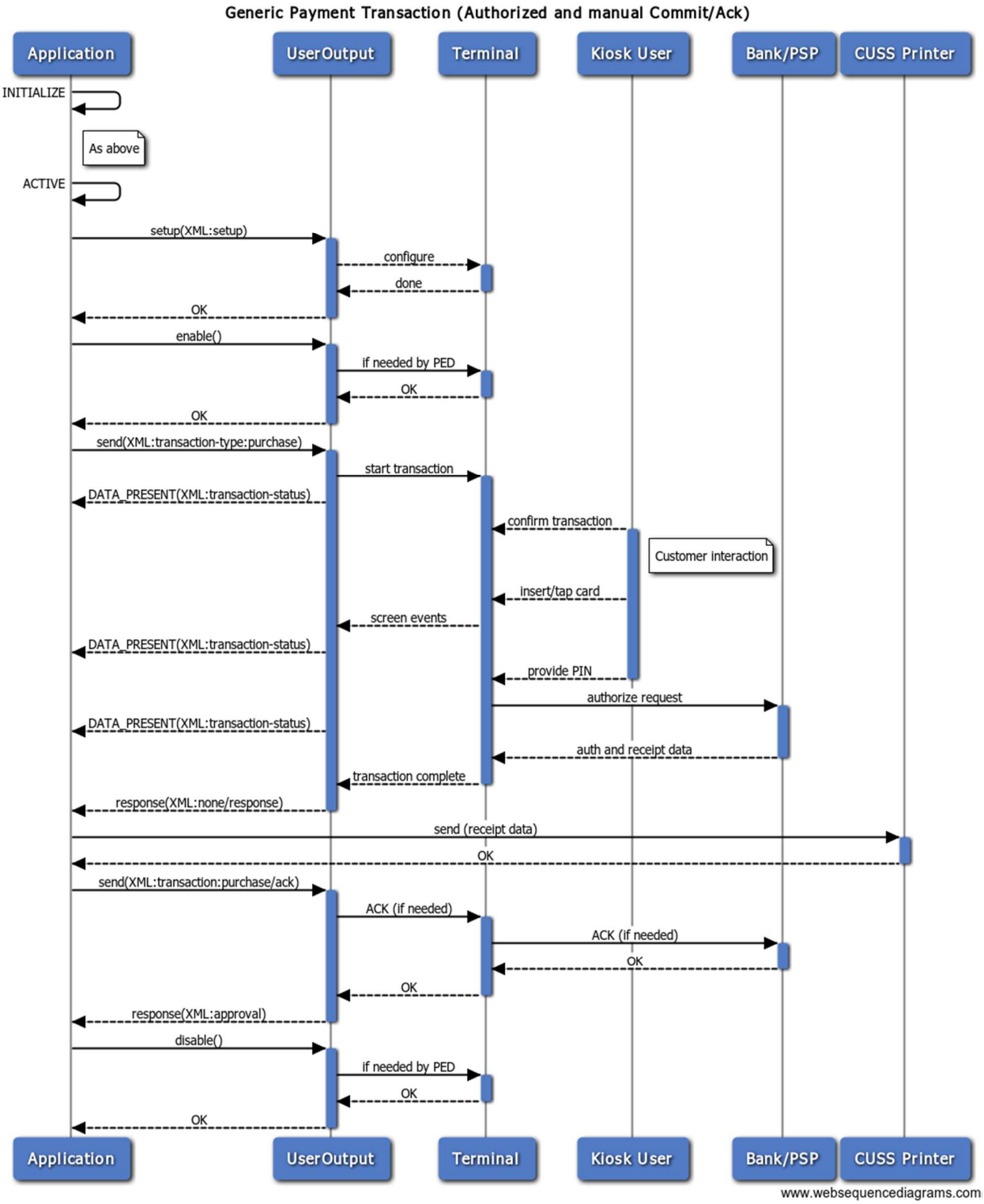
7.19.3 Sequence Diagrams

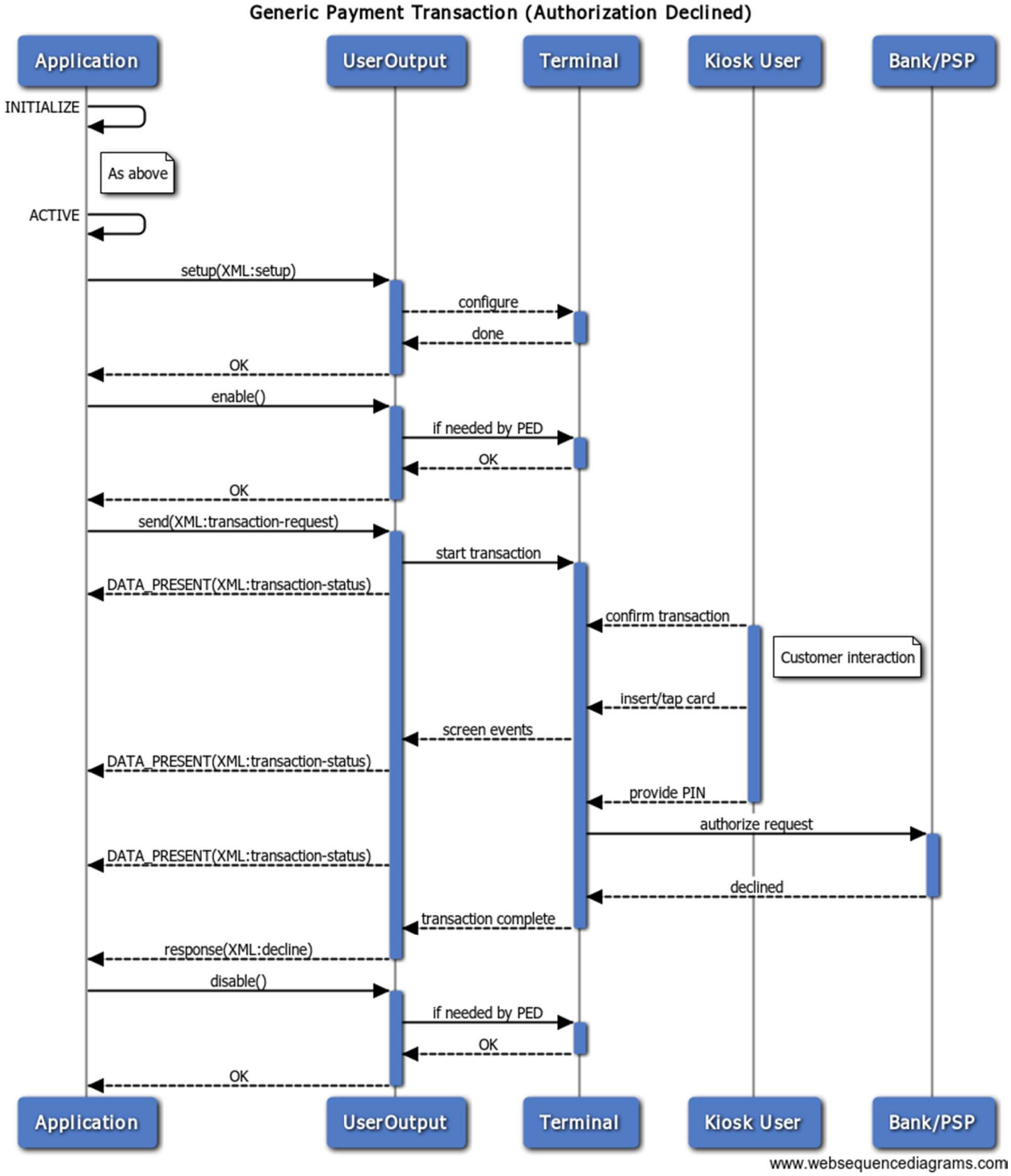
Typical Sequence Diagram for Payment Device:

The directives for the UserOutput component follows the normal CUSS purpose of component directives.

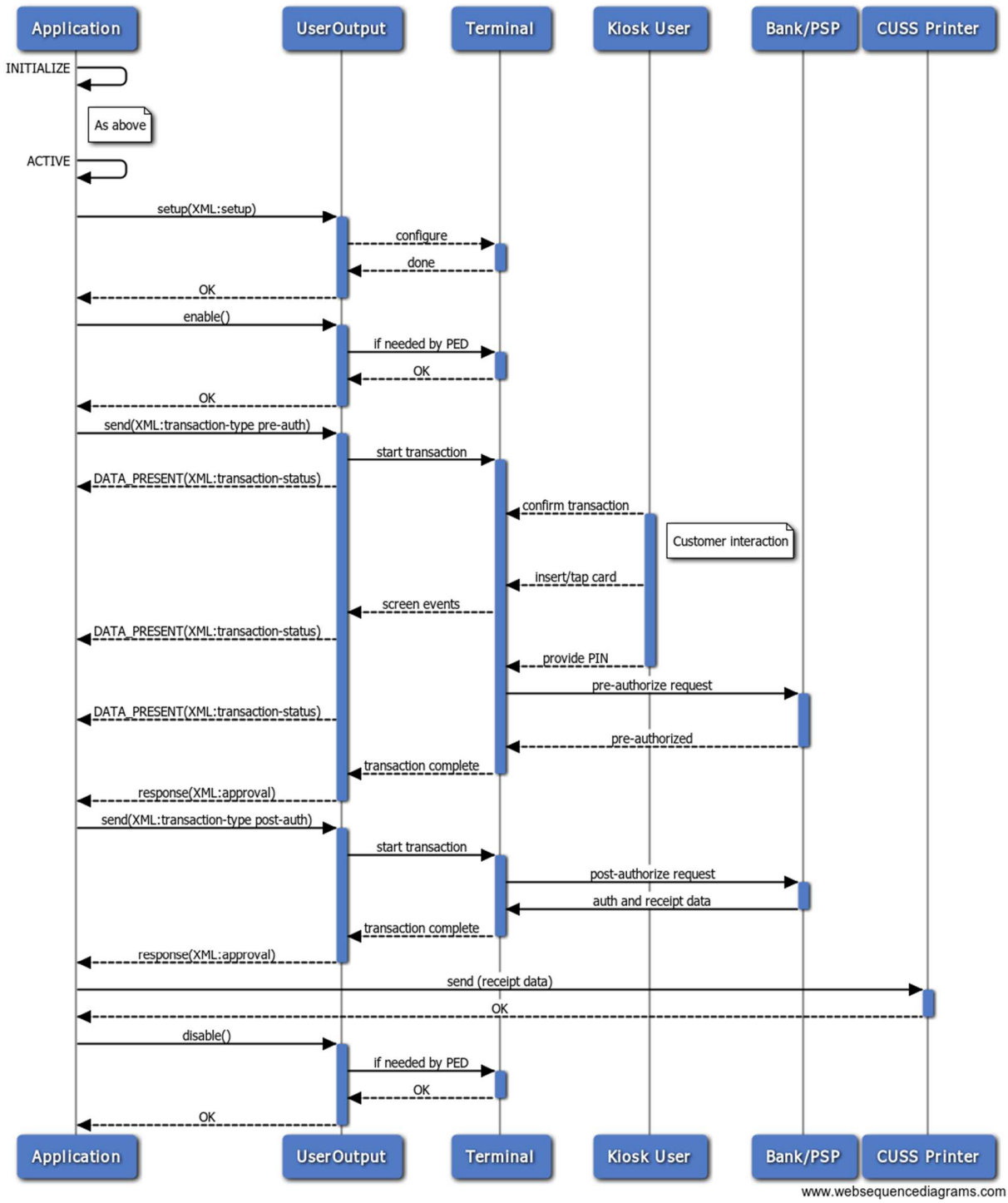
- Review the device Characteristics and firmwareVersion to determine capabilities.
- **acquire()** and **release()** to gain overall access to the component and set up an event listener
- **setup()** for the application to obtain the Characteristics XML message for the device.
- **setup()** for the application to set the component context. The setup data stream must contain an XML message formatted in accordance with the DS_TYPES_EPAYMENT message schema, as defined by the interface file CUSS.PAYMENT.XSD.
- The application must call **setup()** while in the ACTIVE state prior to requesting a payment transaction.
    - As per Section 6.2.1, the configuration provided via the setup() call is in effect until the application returns to the AVAILABLE state, or another call to setup() is made, whichever comes first.
- **enable()** and **disable()** to control when the device is ready for use by the customer. This may be required for some types of devices that require explicit activation, but may not have any effect on other devices.
- **send()** to provide the platform with the terms of the payment transaction and basic itinerary information. The send data stream must contain an XML message formatted in accordance with the DS_TYPES_EPAYMENT message schema, as defined by the interface file CUSS.PAYMENT.XSD.
- The application will need to carry out a payment transaction using one of the payment models supported by the schema:
    - Payment Authorization with automatic Commit
    - Payment Authorization with manual Commit or Cancellation
    - Payment Pre-authorization and Post-Authorization or Cancellation (two-phase)
- The platform will broadcast DATA_PRESENT events to the application's **device listener** for the payment component, containing an XML message formatted in accordance with the DS_TYPES_EPAYMENT message schema, as defined by the interface file CUSS.PAYMENT.XSD, with information on the progress of the transaction (payment terminal screen messages.)
- **cancel()** allows an application to *request* that the current send() request be ended. The platform response to the cancel() request indicates only the status of the cancel request, not whether the transaction was cancelled. To determine if the transaction was in fact cancelled, verify the response to the send() request.
- **query()** and **test()** allow the application to verify the condition of the device at any time, including the state of the component and the health of the payment subsystem


Generic Payment Transaction (Authorized and auto Commit)
Application
UserOutput
Terminal
Kiosk User
Bank/PSP
CUSS Printer
INITIALIZE
Check Characteristics
firmwareVersion: DS_TYPES_EPAYMENT
acquire()
OK
setup()
Get XML:characteristics
OK
ACTIVE
setup(XML:setup)
configure
done
OK
enable()
if needed by PED
OK
OK
send(XML:transaction-type:purchase)
start transaction
DATA_PRESENT(XML:transaction-status)
confirm transaction
Customer interaction
insert/tap card
screen events
DATA_PRESENT(XML:transaction-status)
provide PIN
authorize request
DATA_PRESENT(XML:transaction-status)
auth and receipt data
transaction complete
response(XML:none/response)
send (receipt data)
OK
disable()
if needed by PED
OK
OK
Application
UserOutput
Terminal
Kiosk User
Bank/PSP
CUSS Printer
www.websequencediagrams.com

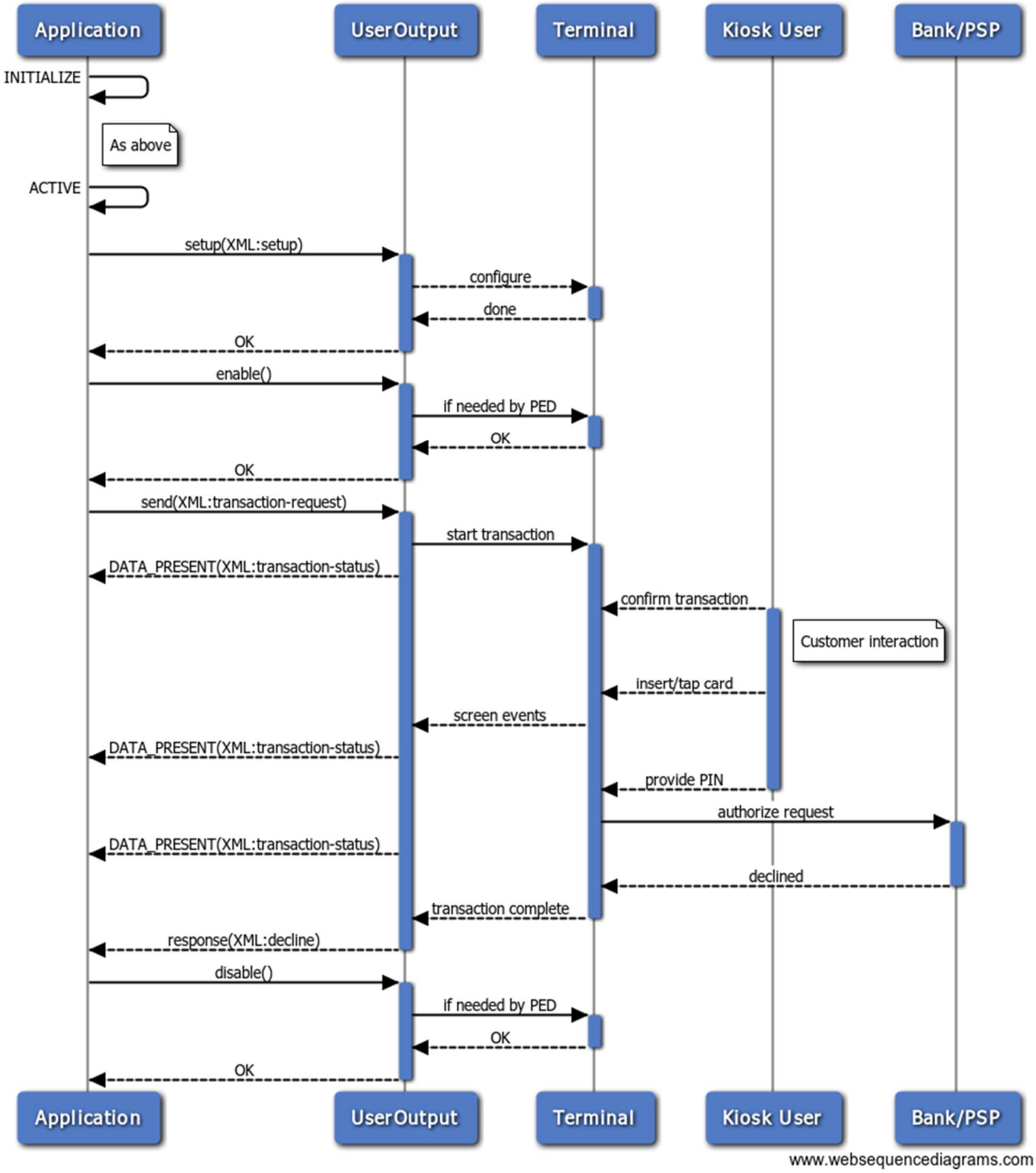

Generic Payment Transaction (Authorized and manual Commit/Ack)
Application
UserOutput
Terminal
Kiosk User
Bank/PSP
CUSS Printer
INITIALIZE
As above
ACTIVE
setup(XML:setup)
configure
done
OK
enable()
if needed by PED
OK
OK
send(XML:transaction-type:purchase)
start transaction
DATA_PRESENT(XML:transaction-status)
confirm transaction
Customer interaction
insert/tap card
screen events
DATA_PRESENT(XML:transaction-status)
provide PIN
authorize request
DATA_PRESENT(XML:transaction-status)
auth and receipt data
transaction complete
response(XML:none/response)
send (receipt data)
OK
send(XML:transaction:purchase/ack)
ACK (if needed)
ACK (if needed)
OK
OK
response(XML:approval)
disable()
if needed by PED
OK
OK
Application
UserOutput
Terminal
Kiosk User
Bank/PSP
CUSS Printer
www.websequencediagrams.com

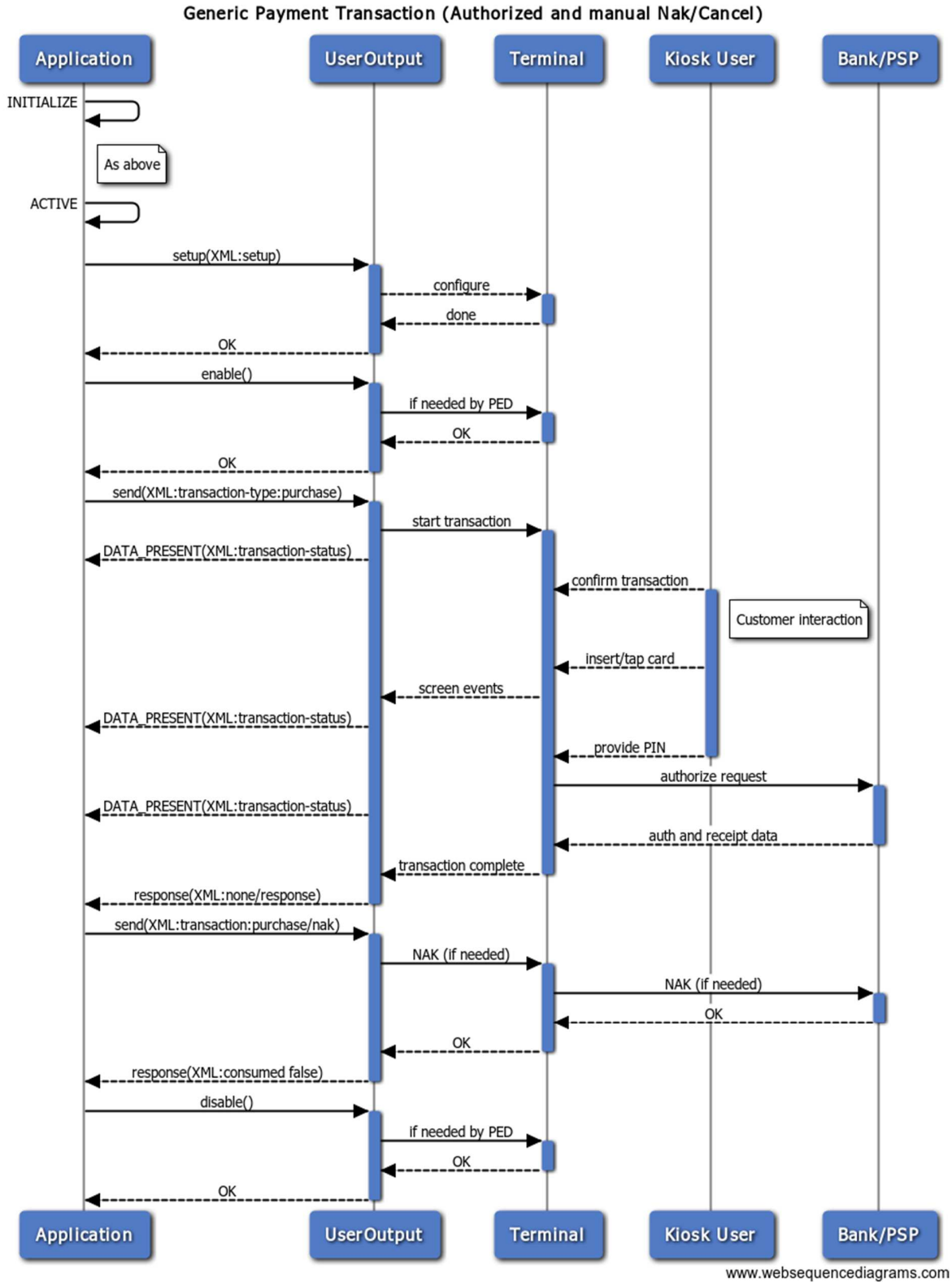

Generic Payment Transaction (Authorization Declined)
Application
UserOutput
Terminal
Kiosk User
Bank/PSP
INITIALIZE
As above
ACTIVE
setup(XML:setup)
configure
done
OK
enable()
if needed by PED
OK
OK
send(XML:transaction-request)
start transaction
DATA_PRESENT(XML:transaction-status)
confirm transaction
Customer interaction
insert/tap card
screen events
DATA_PRESENT(XML:transaction-status)
provide PIN
authorize request
DATA_PRESENT(XML:transaction-status)
declined
transaction complete
response(XML:decline)
disable()
if needed by PED
OK
OK
Application
UserOutput
Terminal
Kiosk User
Bank/PSP
www.websequencediagrams.com

Generic Payment Transaction (Pre-Authorization and Post-Authorization)

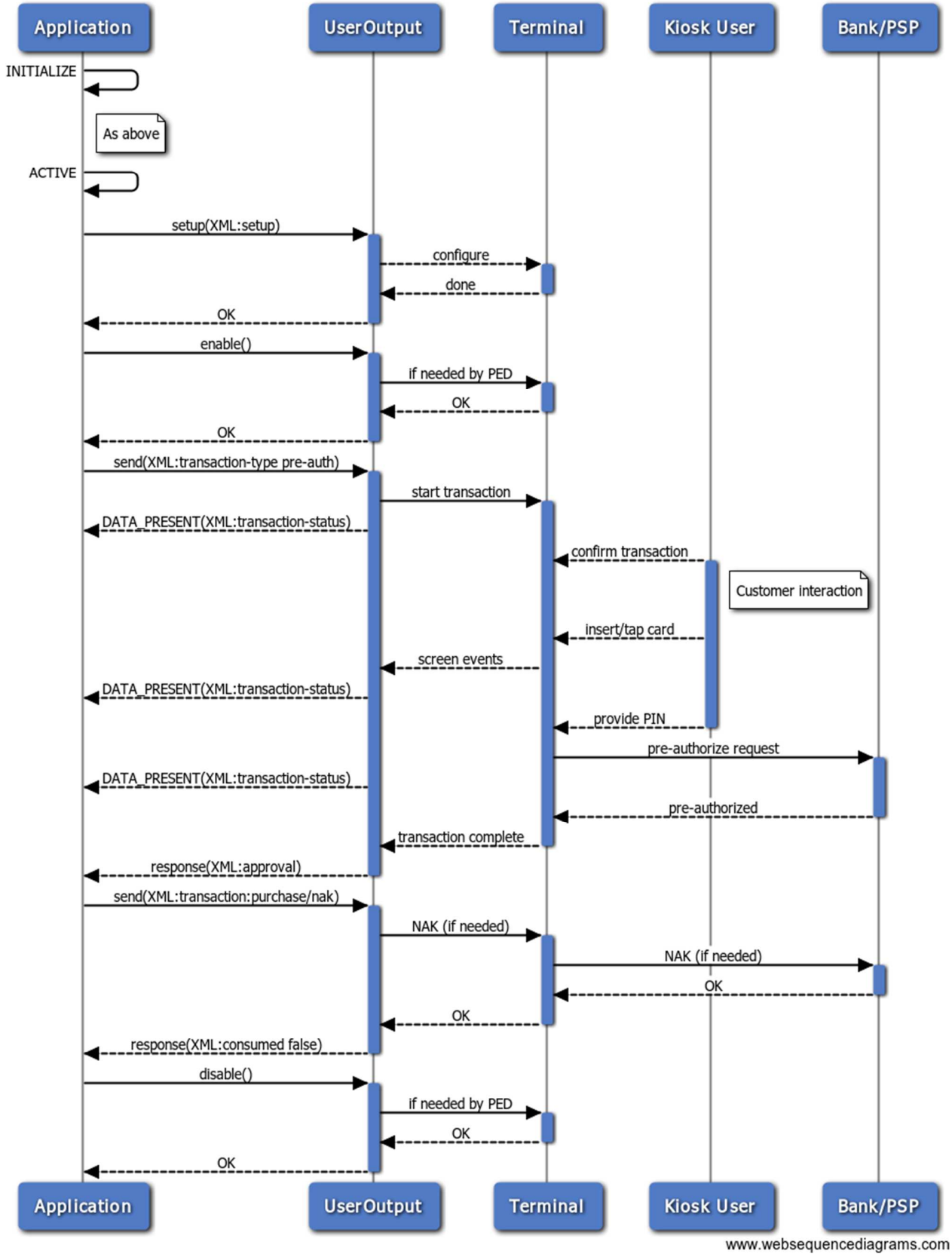


Generic Payment Transaction (Authorization Declined)




Generic Payment Transaction (Authorized and manual Nak/Cancel)
Application
UserOutput
Terminal
Kiosk User
Bank/PSP
INITIALIZE
As above
ACTIVE
setup(XML:setup)
configure
done
OK
enable()
if needed by PED
OK
OK
send(XML:transaction-type:purchase)
start transaction
DATA_PRESENT(XML:transaction-status)
confirm transaction
Customer interaction
insert/tap card
screen events
DATA_PRESENT(XML:transaction-status)
provide PIN
authorize request
DATA_PRESENT(XML:transaction-status)
auth and receipt data
transaction complete
response(XML:none/response)
send(XML:transaction:purchase/nak)
NAK (if needed)
NAK (if needed)
OK
OK
response(XML:consumed false)
disable()
if needed by PED
OK
OK
Application
UserOutput
Terminal
Kiosk User
Bank/PSP
www.websequencediagrams.com

Generic Payment Transaction (Pre-Authorization and cancellation of request)



7.19.4 Explanation of Schema Fields

Here are some clarifications as to the purposes of some message fields defined in the schema.

A CUSS platform must not include any sensitive payment information (in scope for PCI-DSS) in any of the fields/data provided to the CUSS application.

1. ***merchant-id***

   The merchant ID in the CUSS schema is not the same technical value as the mechant ID used by the Payment Service Provider. In CUSS, this optional value lets a CUSS application provide an indicator about which company or provider is to receive the funds for the transaction.

   - The CUSS application provider will need to indicate the list of valid merchant IDs that an application may request at time of payment.
   - The CUSS platform will likely do an internal mapping of application requestvalues, to values cuitable for the platform's payment solution.
   - The CUSS platform will likely validate and filter the merchant-id in transaction requests from the CUSS application.
   - If no value is specified, the platform will use an appropriate value (as set up during initial deployment of payment capabilities) for the transaction based on the companyCode value of the CUSS application making the transaction request.
   - This field is most likely to be used within applications that process transactions for multiple entities, such as ground handling or generic airport applications.

2. ***feature-list***

   This characteristic indicates what modes of payment are supported by the payment terminal. The application requests a particular mode of payment in its setup() request parameters.

   - The auto-commit feature indicates whether or not the application wishes to permorm a manual commit at the end of a successful transaction.
   - The pre-authorization feature indicates whether the pre-authorization/post-authorization payment model is supported

3. ***card-brand-list***

   This characteristic lists all the card brands that can be accepted for payment at this terminal. The application can restrict these types via setup() if the overridable attribute is true.

   - The card-brand values shall match the values used in IATA Resolution 728 (note in particular the use of "CA" for Mastercard in this Resolution)

4. ***media-type-list***

   This characteristic lists all media types that the payment terminal is able to accept. The application can restrict these types via setup() if the overridable attribute is true.

   - The media-type values are as restricted in the XSD, being at time of publication one of:
     - icc
     - mag-stripe
     - contactless

5. currency-code-list

This characteristic lists all currency codes that the payment terminal is able to select. An application must indicate one of these codes when making a request for a payment transaction.

- The currency-code values shall match the currency codes defined by ISO4217 but shall not be case sensitive.

6. *overridable*

This attribute indicates if the payment feature, such as card-brand, can be overridden by the application using a setup() request.

7. environment

The values in the environment block are optional, and allow the CUSS application to indicate more specific values related to the payment transaction, for tracking and reconciliation purposes.

- If the CUSS application does not supply any values, the CUSS platform may provide appropriate values to the payment subsystem depending on what data fields that subsystem requires.

8. epayment-msg-id

This value is an arbitrary tracking value that the CUSS application chooses and assigns when starting a payment transaction. The CUSS platform must then echo this requested value in all subsequent responses or asynchronous event messages related to that payment transaction.

- This approach is to permit a CUSS application architecture that matches and associates messages.
- This capability is critical for the potential case where an application has more than one multi-step payment transaction (manual ack, or auth/hold/commit) ongoing at once.

9. ***transaction-request-language***

This is an optional value that lets the CUSS application suggest a language to use when performing the payment transaction. The payment subsystem could use this value to adjust prompts displayed on the payment terminal screen or the language on the preformatted receipt data.

- This value could be set in response to a prior language selection by the kiosk application user as part of the overall kiosk transaction.
- The payment system may override this setting once a language value is read from the card itself as part of the transaction.

10. ***itinerary***

The information in the itinerary block is optional, and lets the CUSS application specific additional tracking and reconciliation data for the payment transaction, relating specifically to an airline transaction itinerary.

- The CUSS platform provider may mandate that this information be provided with all transactions. This requirement will be established during the negotiation and configuration phase of the initial deployment of payment capability for an airline.
- It is anticipated that the CUSS platform itself will not store or make use of this information. Instead, it will be forwarded to the aggregator, refund, and tracking components of a common use payment solution infrastructure.

11. ***billing-description***

This optional value lets the CUSS application suggest a value that will appear on the kiosk application user's payment summary (monthly bill, etc.) as an extended description of the transaction.

- The value may or may not appear on the customer's bill depending on the capabilities of the airport and customer's banking systems.
- Long data may be truncated
- The platform may also be used as an additional tracking mechanism for other systems such as aggregator and reconciliation infrastructure

12. ***reference***

This optional value allows the CUSS application to specify and to receive arbitrary reference or reconciliation data from the payment interface.

- How this is used will vary from provider to provider and is likely related to the aggrement the application supplier has with their acquiring bank or payment aggregator.

13. ***approval***

This element describes the specific information provided as response to an approved payment transaction.

- An application may use this information as it decides
- No sensitive payment information shall be provided in this information
- Various data elements, such as *cryptogram,* may be required by some application providers for proper payment processing and reconciliation

14. ***form-of-payment-id***

This element is a identifier that references the payment transaction uniquely across all card brands, and can be used by airline applications for reconciliation, tracking and refund purposes.

- It may or not be available, depending on the capabilities and software version of the payment solution used

15. ***non-approval***

This element describes the specific reason a transaction was not approved.

- The non-approval-reason-code and referral values are not standard and will vary between payment systems

- These values are likely useful for logging and reconciliation purposes, and would not likely be exposed to the kiosk user or affect the business logic of the kiosk transaction.

16. ***receipt-data***

This element provides pre-formatted receipt data that is the application's responsibility to print. The data format is in raw text including line feeds, and may be encoded as a CDATA element.

- It is an application business logic decision, and potentially a bank certification requirement, to properly format and print this data on a document for the customer.
- Key data elements used on the receipt are also proved separately under the *approval* response.

17. ***transaction-document-return-type***

This attribute of the transaction field indicates the type of follow up message that the payment system expects. In particular, this is used to indicate if the application must send a transaction-ack message to consume an authorized transaction amount.

18. ***gp-parameter-list***

This is a list of special or proprietary payment parameters that can be provided by the CUSS application when requesting the payment, and returned by the payment subsystem upon conclusion of the transaction. These are optional values.

- This mechanism is included in the schema to allow certain legacy payment protocols to be migrated to use the new CUSS 1.3 payment interface
- It is also included to allow for flexibility for future payment industry requirements that may affect common use or airline payment.

7.19.5 Example Schema Messages

The examples shown in this section may not accurately reflect the most recent version of the CUSS.PAYMENT.XSD. Always refer to the schema definition for message creation and validation.

Obtain Characteristics and Capabilities (application INITIALIZE)

By reading the XML message contents of the CUSS payment component's firmwareVersion field, review the capabilities of the application.

*Sample platform **firmwareVersion characteristics** message:*

```
<?xml version="1.0" encoding="UTF-8"?>
<cu-msg xsi:noNamespaceSchemaLocation="CuMsgSchema.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" cu-msg-document-
type="characteristics" cu-msg-schema-version="1.00">
  - <characteristics ds-types="DS_TYPES_EPAYMENT">
      - <epayment>
          - <feature-list>
                <feature overridable="true" feature-name="auto-commit">false</feature>
                <feature feature-name="pre-authorization">true</feature>
            </feature-list>
          - <card-brand-list overridable="true">
                <card-brand>amex</card-brand>
                <card-brand>mastercard</card-brand>
                <card-brand>visa</card-brand>
            </card-brand-list>
          - <media-type-list overridable="true">
                <media-type>icc</media-type>
                <media-type>mag-stripe</media-type>
            </media-type-list>
          - <currency-code-list>
                <currency-code>chf</currency-code>
                <currency-code>eur</currency-code>
            </currency-code-list>
        </epayment>
    </characteristics>
</cu-msg>
```

Configure to only accept certain types of payment (application INITIALIZE/ACTIVE setup() request)

Request that the platform restrict the types of payment that are accepted. Depending on the technical capabilities of the solution, the platform may or may not be able to honour the request.

Sample application request message:

```
<?xml version="1.0" encoding="UTF-8"?>
<cu-msg xsi:noNamespaceSchemaLocation="CuMsgSchema.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" cu-msg-document-
type="epayment-msg" cu-msg-schema-version="1.00">
  - <epayment-msg epayment-msg-type="setup" epayment-msg-id="A12345">
      - <setup>
            <merchant-id>AA</merchant-id>
          - <feature-list>
                <feature feature-name="auto-commit">true</feature>
            </feature-list>
          - <card-brand-list>
                <card-brand>mastercard</card-brand>
                <card-brand>visa</card-brand>
            </card-brand-list>
          - <media-type-list>
                <media-type>icc</media-type>
                <media-type>mag-stripe</media-type>
            </media-type-list>
        </setup>
    </epayment-msg>
</cu-msg>
```

Submit a payment request for an immediate transaction (application ACTIVE send() request)

Request that the terminal carry out a payment transaction and immediately commit the transaction once the payment is complete.

*Sample application **request** message:*

```
<?xml version="1.0" encoding="UTF-8"?>
<cu-msg xsi:noNamespaceSchemaLocation="CuMsgSchema.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" cu-msg-document-
type="epayment-msg" cu-msg-schema-version="1.00">
  - <epayment-msg epayment-msg-type="transaction" epayment-msg-id="A12345">
      - <transaction transaction-type="purchase" transaction-document-return-type="response"
        transaction-document-type="request">
          - <transaction-request language="en">
              - <environment>
                    <merchant-id>AA</merchant-id>
                    <location>Kiosk-AA36</location>
                    <cashier>Kiosk</cashier>
                </environment>
              - <itinerary>
                    <name>JOHN CORRICK</name>
                    <pnr>GMTFOH</pnr>
                    <date>23SEP2013</date>
                    <flight-number>DL1713</flight-number>
                    <departure-city>MCO</departure-city>
                    <arrival-city>LGW</arrival-city>
                </itinerary>
                <billing-description>DELTA FLT MCO-LGW 092312</billing-description>
              - <authorization-amounts>
                  - <requested-amounts currency-code="usd">
                        <base-amount>100.00</base-amount>
                    </requested-amounts>
                </authorization-amounts>
            </transaction-request>
        </transaction>
    </epayment-msg>
</cu-msg>
```

Submit a payment hold for an amount to be confirmed later (application ACTIVE send() request)

The application, in anticipation of an unknown final amount, requested preauthorization of transaction amount.

*Sample application **request** message:*

```
<?xml version="1.0" encoding="UTF-8"?>
<cu-msg xsi:noNamespaceSchemaLocation="CuMsgSchema.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" cu-msg-document-
type="epayment-msg" cu-msg-schema-version="1.00">
  - <epayment-msg epayment-msg-type="transaction" epayment-msg-id="A12345">
      - <transaction transaction-type="pre-auth" transaction-document-return-type="response"
        transaction-document-type="request">
          - <transaction-request language="en">
              - <environment>
                    <merchant-id>AA</merchant-id>
                    <location>Kiosk-AA36</location>
                    <cashier>Kiosk</cashier>
                </environment>
              - <authorization-amounts>
                  - <requested-amounts currency-code="usd">
                        <base-amount>100.00</base-amount>
                    </requested-amounts>
                </authorization-amounts>
            </transaction-request>
        </transaction>
    </epayment-msg>
</cu-msg>
```

Track the progress of the payment transaction (application ACTIVE status DATA_PRESENT events)

The platform will send progress updates for the transaction based on prompts provided to the user, via unsolicited status messages:

*Sample platform **asynchronous status** messages:*

```
<?xml version="1.0" encoding="UTF-8"?>
<cu-msg xsi:noNamespaceSchemaLocation="CuMsgSchema.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" cu-msg-document-
type="epayment-msg" cu-msg-schema-version="1.00">
  - <epayment-msg epayment-msg-type="transaction" epayment-msg-id="A12345">
      - <transaction transaction-type="purchase" transaction-document-return-type="none"
        transaction-document-type="response">
          - <transaction-response response-type="status">
                <transaction-status transaction-status-id="card-insertion">INSERT
                    CARD</transaction-status>
            </transaction-response>
        </transaction>
    </epayment-msg>
</cu-msg>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<cu-msg xsi:noNamespaceSchemaLocation="CuMsgSchema.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" cu-msg-document-
type="epayment-msg" cu-msg-schema-version="1.00">
  - <epayment-msg epayment-msg-type="transaction" epayment-msg-id="A12345">
      - <transaction transaction-type="purchase" transaction-document-return-type="none"
        transaction-document-type="response">
          - <transaction-response response-type="status">
              <transaction-status transaction-status-id="pin-entry">ENTER PIN</transaction-
                status>
            </transaction-response>
        </transaction>
    </epayment-msg>
</cu-msg>

<?xml version="1.0" encoding="UTF-8"?>
<cu-msg xsi:noNamespaceSchemaLocation="CuMsgSchema.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" cu-msg-document-
type="epayment-msg" cu-msg-schema-version="1.00">
  - <epayment-msg epayment-msg-type="transaction" epayment-msg-id="A12345">
      - <transaction transaction-type="purchase" transaction-document-return-type="none"
        transaction-document-type="response">
          - <transaction-response response-type="status">
              <transaction-status transaction-status-id="processing">PROCESSING... PLEASE
                WAIT</transaction-status>
            </transaction-response>
        </transaction>
    </epayment-msg>
</cu-msg>

<?xml version="1.0" encoding="UTF-8"?>
<cu-msg xsi:noNamespaceSchemaLocation="CuMsgSchema.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" cu-msg-document-
type="epayment-msg" cu-msg-schema-version="1.00">
  - <epayment-msg epayment-msg-type="transaction" epayment-msg-id="A12345">
      - <transaction transaction-type="purchase" transaction-document-return-type="none"
        transaction-document-type="response">
          - <transaction-response response-type="status">
              <transaction-status transaction-status-id="card-removal">APPROVED, PLEASE
                REMOVE YOUR CARD</transaction-status>
            </transaction-response>
        </transaction>
    </epayment-msg>
</cu-msg>
```

Receive the Approval for a successful payment transaction (response to send() request)

The platform will send progress updates for the transaction based on prompts provided to the user, via unsolicited status messages:

*Sample platform **response** message:*

```
<?xml version="1.0" encoding="UTF-8"?>
<cu-msg xsi:noNamespaceSchemaLocation="CuMsgSchema.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" cu-msg-document-
type="epayment-msg" cu-msg-schema-version="1.00">
  - <epayment-msg epayment-msg-type="transaction" epayment-msg-id="A12345">
      - <transaction transaction-type="purchase" transaction-document-return-type="ack"
        transaction-document-type="response">
          - <transaction-response response-type="approval">
              - <approval>
                    <transaction-reference>87654321</transaction-reference>
                    <media-type>icc</media-type>
                    <approval-type>pin</approval-type>
                    <card-brand>visa</card-brand>
                    <card-pan>401173XXXXXX5218</card-pan>
                    <approval-code>123456</approval-code>
                  - <gp-parameter-list>
                        <gp-parameter gp-parameter-name="retrieval-reference-
                            number">642618</gp-parameter>
                    </gp-parameter-list>
                  - <authorization-amounts>
                      - <approved-amounts currency-code="usd">
                            <base-amount>100.00</base-amount>
                        </approved-amounts>
                    </authorization-amounts>
                    <receipt-data> APPROVED: 123456 THANK YOU!!! </receipt-data>
                </approval>
            </transaction-response>
        </transaction>
    </epayment-msg>
</cu-msg>
```

Receive a Decline for a rejected payment transaction (response to send() request)

The platform will send progress updates for the transaction based on prompts provided to the user, via unsolicited status messages:

*Sample platform **response** message:*

```
<?xml version="1.0" encoding="UTF-8"?>
<cu-msg xsi:noNamespaceSchemaLocation="CuMsgSchema.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" cu-msg-document-
type="epayment-msg" cu-msg-schema-version="1.00">
  - <epayment-msg epayment-msg-type="transaction" epayment-msg-id="A12345">
      - <transaction transaction-type="purchase" transaction-document-return-type="none"
        transaction-document-type="response">
          - <transaction-response response-type="decline">
              <non-approval referral="true" non-approval-reason-code="5566">Please
                  Call</non-approval>
            </transaction-response>
        </transaction>
    </epayment-msg>
</cu-msg>
```

Confirm/Ack the consumption of the approved amount (application ACTIVE send() request)

The application confirms the consumption/use of the approved amountThe platform will send progress updates for the transaction based on prompts provided to the user, via unsolicited status messages. The epayment-msg-id can be used to associate the acknowledgement with a previous request.:

*Sample application **request** message:*

```
<?xml version="1.0" encoding="UTF-8"?>
<cu-msg xsi:noNamespaceSchemaLocation="CuMsgSchema.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" cu-msg-document-
type="epayment-msg" cu-msg-schema-version="1.00">
  - <epayment-msg epayment-msg-type="transaction" epayment-msg-id="A12345">
      - <transaction transaction-type="purchase" transaction-document-return-type="none"
        transaction-document-type="ack">
            <transaction-ack consumed="true"/>
        </transaction>
    </epayment-msg>
</cu-msg>
```

Finalize a transaction amount after a pre-authorization (application ACTIVE send() request)

Having a successful pre-authorization on hand, post-authorize and consume an amount less than or equal to the preauthorization amount.

*Sample application **request** message:*

```
<?xml version="1.0" encoding="UTF-8"?>
<cu-msg xsi:noNamespaceSchemaLocation="CuMsgSchema.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" cu-msg-document-
type="epayment-msg" cu-msg-schema-version="1.00">
  - <epayment-msg epayment-msg-type="transaction" epayment-msg-id="A12345">
      - <transaction transaction-type="post-auth" transaction-document-return-
        type="response" transaction-document-type="request">
          - <transaction-request language="en">
              - <environment>
                    <merchant-id>AA</merchant-id>
                    <location>Kiosk-AA36</location>
                    <cashier>Kiosk</cashier>
                </environment>
              - <pre-auth>
                    <transaction-reference>87654321</transaction-reference>
                    <approval-code>123456</approval-code>
                  - <approved-amounts currency-code="usd">
                        <base-amount>100.00</base-amount>
                    </approved-amounts>
                </pre-auth>
              - <itinerary>
                    <name>JOHN CORRICK</name>
                    <pnr>GMTFOH</pnr>
                    <date>23SEP2013</date>
                    <flight-number>DL1713</flight-number>
                    <departure-city>MCO</departure-city>
                    <arrival-city>LGW</arrival-city>
                </itinerary>
                <billing-description>DELTA FLT MCO-LGW 092312</billing-description>
              - <authorization-amounts>
                  - <requested-amounts currency-code="usd">
                        <base-amount>75.00</base-amount>
                    </requested-amounts>
                </authorization-amounts>
            </transaction-request>
        </transaction>
    </epayment-msg>
</cu-msg>
```

7.19.6 Non-Payment Magnetic Card Support

To support a generic payment interface, a kiosk enclosure will need a card reader to perform payment transactions.

To support traditional magnetic card transactions such as form of identification (FOID), and legacy payment transactions that do not use the generic payment interface, a kiosk enclosure will need a card reader that can read raw payment and FOID track data.

The CUSS Technical Specification has not removed support for generic card reading in version 1.3, and there is no explicit timeline in place to remove this capability in the future.

However, for reasons of customer usability and ease of use, as well as simplified kiosk enclosure design and certification, it is desirable to have a single physical card reader that accomplishes both tasks.

Some kiosk enclosures will not be able to meet this goal, and will be deployed with two physical card readers. In this case, the CUSS platform will present two interfaces:

- A generic payment interface as defined in this section which does not include a MediaInput component
- A direct card reader interface for FOID and legacy payment transactions, configured as defined in Section 7.7 or 7.8

Other kiosks enclosures will incorporate a single card reader that can serve both purposes. In this case, the CUSS platform will present a combined interface:

- A generic payment interface as defined in this section which also includes a MediaInput component
- There will be no separate card reader components
- The MediaInput component of the generic payment device will behave as described in Section 7.8
- Existing applications will continue to be able to find and use the direct card reader interface using the mechanisms described in Section 7.8, without change, including FOID, DISCRETIONARY, and PAYMENT modes.

There are some special CUSS platform considerations when running a combined purpose single card reader:

- The payment solution provider and payment terminal provider may need to deploy specialized software or configuration in order to read cards as described in Section 7.8.
- The generic card reader feature must implement the FOID, DISCRETIONARY and PAYMENT detection and truncation rules as described within this specification.
- For PCI-DSS scope reasons, it may be preferable to have these rules implemented in the certified payment terminal software.

- There may be additional certification requirements to deploy these changes to the payment terminal and card reader controller.

- The platform must ensure that generic card reader, and generic payment transactions cannot operate at the same time, by preventing applications from calling enable() on both components at the same time.

- If a kiosk enclosure includes a single card reader device for both purposes that is *motorized* then instead of a single MediaInput component, it shall be represented by the correct multiple components defined in Section 7.7 for a motorized reader.

7.20 RFID and e-Passport Readers

Description of Device:

The CUSS Technical Specification 1.3 does not include any control interfaces for RFID-capable e-Passport readers.

7.21 Accessible Kiosk Interfaces

Description of Device:

The CUSS Technical Specification 1.3 does not define any interface to allow facilitating devices on Accessible Kiosks, such as navigation keypads, audio feedback and headset control.

Ch 8: FOID and Payment Card Handling

This chapter is based on the CUSS FOID Addendum published by IATA in June 2011. All CUSS systems, regardless of version of the CUSS Technical Specification, must have implemented the CUSS FOID Addendum on or after June 30th 2012 to remain CUSS-compliant.

The separate CUSS FOID Addendum document applies to all CUSS 1.0, CUSS 1.1 and CUSS 1.2 implementations. This CUSS 1.3 Technical Specification chapter supersedes the CUSS FOID Addendum document for all CUSS 1.3 site implementations.

8.1 Introduction and Summary

This chapter defines how the CUSS specification restricts how CUSS applications read payment card data from the kiosk. Card issuer operational regulations state:

> *A Merchant must not request or use an Account Number for any purpose other than as payment for goods and services.*

The CUSS 1.3 separates access to card data between two types. First, CUSS applications retain full access to the card data in cases where it is needed for payment processing. Second, for all card transactions that are not for payment, the content of the payment card is truncated (obscured) so that it retains essentially information such as the name, but no longer has an Account Number and is hence not considered a payment card.

> **This change to the CUSS specification is critical for card issuers, as a condition for continuing to accept payments at airline self-service kiosks. As stated in the amended IATA Recommended Practice RP1706c, all CUSS 1.0, CUSS 1.1 and CUSS 1.2 must implement the changes described in the original CUSS FOID Addendum by June 30th, 2012.**
>
> **All CUSS 1.3 sites, whenever they are deployed, must abide by the terms if this chapter, which is a superset of the original CUSS FOID Addendum guidance.**

The CUSS Technical Specification is changed to:

- Define detection rules to identify when a card read at a kiosk is a payment card
- Define data modification rules to indicate how card track data is truncated to remove sensitive data
- Add new Extended Media Types that applications must use to access full payment card data

- Change the default behaviour of the existing card ISO data type to return truncated payment card data
- Set a date, via IATA RP1706c, by which time this Addendum must be deployed in order for a CUSS site to comply with the industry Recommended Practice

The CUSS Technical Specifications changes allow existing CUSS applications to continue to operate and read cards without requiring any applications changes. However, unless modified to support these changes, CUSS applications will not have access to full payment card data and will not be able to process payments based on magnetic cards – the platform will only provide truncated card data.

8.2 Definitions and Goals

CUSS self-service kiosk applications use magnetic card track data for two types of operation:

1. Extract name and account number to perform a financial payment transaction

2. Read card information to use as part of a transaction that is not financial. Typically, this includes using a magnetic card as a form of identification and other tasks, such as Frequent Traveler number transactions.

In short, the two types of transaction are **PAYMENT transactions**, and all other transactions (for brevity, all non-payment transactions) are considered “form of identification”, or **FOID transactions**. During a payment transaction, a CUSS application reads, processes and transmits the Payment Account Number (PAN) data, and is subject to data security controls including provisions of the PCI-DSS (Payment Card Industry – Data Security Standard).

This chapter is a mandatory component of the CUSS Technical Specification 1.3 for card reader components, in order to abide by Card Brands Operating Regulations (regarding FOID not using PAN) and facilitate data security compliance efforts for CUSS platforms and applications. The specific goals of the change are to:

1. Separate card reader data read requests into two separate types. One specific request for access to payment data, and a different request for FOID. This defines access to payment card data as a separate, intentional request in the CUSS platform.

2. Allow CUSS platforms and CUSS applications to separate and modularize the card reading logic within their application architecture, which can isolate payment data protection to more specified, well-defined “need to know” areas and transactions into specific & isolated software components.

3. Remove all sensitive payment information from card data sent to CUSS applications for FOID transactions. This data truncation prevents any exposure to sensitive payment data in those parts of the application that perform FOID card transactions.

4. Establish a firm cutover date by which all CUSS platforms must implement this chapter in order to remain compliant with the CUSS Technical Specification, including retroactive compliance with CUSS-TS 1.0, 1.1 and 1.2. CUSS applications which perform payment transactions using the card reader must also implement the change in this chapter by the cutover data, to continue processing payment. CUSS applications that do not change will no longer have access to payment card data.

Scenario without this Chapter:

Every CUSS application that activates and reads magnetic cards, for whatever purpose, always receives full card data for every request, including sensitive payment card data.

Hence any CUSS application that uses magnetic cards in any form is subject to PCI-DSS, whether or not payments are processed.

For this reason, full PCI-compliance of the common use kiosk solution extends beyond the platform and includes the necessary compliance of each CUSS application running on the kiosk as well as any/all network segments or appliances, be it over shared or private networks.

Scenario with the changes in this Chapter:

CUSS applications can continue using magnetic cards, but all sensitive payment data is first truncated by the CUSS platform.

Applications that still do need the payment data must now make an explicit request to the platform. In turn, the platform can decide whether or not a request for payment data is honoured, without affecting other card transactions or applications.

This allows application and platform providers much better control (in architecture, as well as operationally) over access to payment card data on a "need to know" basis.

This helps define more accurately the scope of PCI-DSS as it relates to the common use kiosk and assists with limiting the compliance efforts to a smaller subset (or PCI footprint) of specific payment components.

This chapter applies to all card reader input (read) devices, including dip, swipe and motorized readers, as well as the reader component of card encoders. It does not affect other input or output devices, such as passport or barcode scanners, or card writer interfaces.

8.3 Payment Data Card Definition

This chapter addresses only the following specific industry payment card formats:

1. Magnetic cards encoded in accordance with the ***ISO/IEC 7811 Identification cards — Recording technique*** with track data in accordance with ***ISO/IEC 7813 Identification cards - - Financial transaction cards***

2. **JIS-I** magnetic stripe on back of card with data format equivalent to ***ISO/IEC 7813***

3. Magnetic encoding in accordance with **ANSI X4.16**

4. **JIS-II** magnetic stripe on front of card, as defined by ***JIS X 6302 Type 2***

The goal of this chapter is to redefine how CUSS applications access the complete track data for standard magnetic payment cards. This data is considered sensitive under PCI-DSS and other data security guidelines. These changes make compliance efforts for CUSS platforms and CUSS applications easier, by:

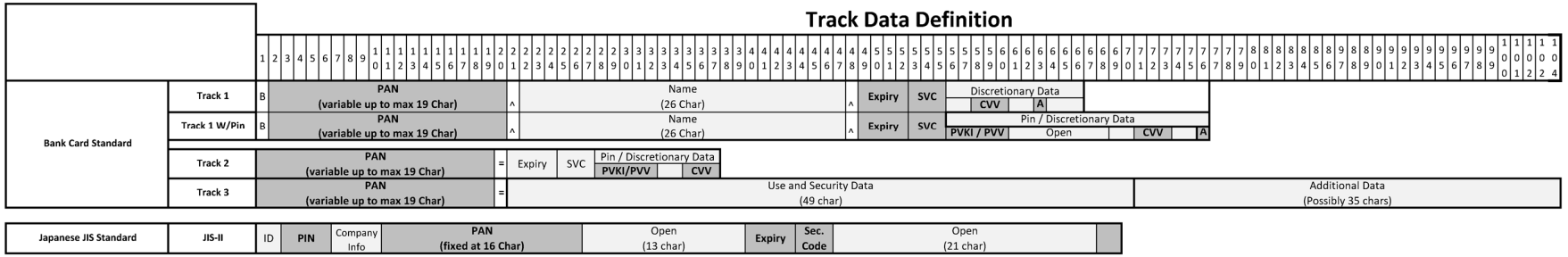
1. Separating access to magnetic card data into two distinct types: full access for payment purposes, and access for purposes of identification and other non-payment transactions. This allows **segregated access** on a “need to know” bases within CUSS applications that read cards.

2. This segregation allows modularization with the CUSS applications and platforms which allows the access to the full payment card to be isolated within dedicated modules. These **isolated modules** can then be specifically controlled within the scope of PCI-DSS, while other non-payment modules fall out of scope.

3. Different access methods for payment and non-payment card data access allows a CUSS application to **completely opt out of receiving sensitive data**, yet still process cards (such as Frequent Traveller cards, or payment cards for identification purposes only.) Likewise, it provides a mechanism for CUSS platforms to **restrict or deny access to payment card data** (without affecting access to non-payment data), providing more control over the security and data exposure on the kiosks.

With these three key points, the changes to the CUSS Technical Specification allow CUSS application and platform providers to better control how sensitive payment card data is accessed on the kiosk, without affecting other magnetic card processing. This helps properly define and limit what is in scope for PCI-DSS compliance and other security measures.

The table on the next page indicates the sensitive data that occurs in payment card track data, as defined in ***ISO/IEC 7813 Identification cards -- Financial transaction cards or the ANSI X4.16 standard***. For more information on payment card data layout, please refer to the ISO/IEC or ANSI specifications and to the *Payment Card Track Data Tutorial* available from the PCI Security Standards Council. Processing of data segments highlighted in Red (see Track Data Definition table) is in scope for PCI-DSS compliance.



- **PAN** – Payment Account Number as defined by ***ISO/IEC 7812 or ANSI X4.16 .*** The first characters of the PAN are the ***IIN (Issuer Identification Number)*** for that PAN.
- **Expiry Date** – Payment Card Expiry Date (4 characters)
- **SVC** – Payment Card Service Code (3 characters)
  Note:
  ANSI X4.16 specifies a validity date of 4 characters in this area followed by 5 characters of discretionary data.
- **CVV** – Card Verification Value or similar other security code
- **A** (**ACI)** – Authorisation Control Index
- **PIN, PVKI, PVV** – Personal Identification Number or similar private key
- **Discretionary, Open** – Discretionary Data, possibly including sensitive data
- **^** – Track one separator character as listed in **ISO/IEC 7813**
- **=** – Track two separator character as listed in **ISO/IEC 7813**[44]

[44] This Chapter does not address ISO7816. For reference, under ISO7816, the separator is 0x0D retrieved from the EMV Track 2 Equivalent Data Element held on chip card (sometimes referred to as the Magnetic Stripe Image or MSI).

8.4 Payment Data Truncation Rules and Requirements

As part of the change to the CUSS specification described below, platforms will need to truncate the track data for payment cards read in the kiosk card reader. For the CUSS Technical Specification, the term truncation is as defined in the PCI-DSS:

> *Practice of removing data segment. Commonly, when account numbers are truncated, the first 12 digits are deleted, leaving only the last 4 digits*

Within the PCI-DSS, the term "truncation" applies the data that is stored or transmitted, whereas the term "masking" applies to data that is visible on screen or on printed documents. More technically speaking, in the context of CUSS magnetic card track data, truncation is:

> *Replacement of some or all characters positions within a stream of characters with a defined, specific neutral character, to permanently obscure some or all data within the character stream with neutral data.*

Data truncation applies to all card track data that represent sensitive Payment Card track data. A card is considered a Payment Card if all the following conditions are met:

1. If Track 1 is present, then on track 1 **all the following rules apply**
    a. The length of the PAN is calculated by counting all non-space characters in the PAN field (ie, spaces are allowed, but excluded from the PAN length calculation)
        i. The length of PAN is greater than or equal to 12, and
        ii. The PAN area contains only numeral data and spaces, and
        iii. The PAN IIN prefix is not listed in the Platform Truncation Exclusion List (see below)
    b. The field separator is '^', and
        i. The track must contain a minimum of two separators
    c. The length of the Expiry Code(in yymm format) is equal to 4, and
        i. The Expiry Date contains only numeric data
        ii. The numeric value of the 'mm' subfield is between 1 and 12
    d. The length of the SVC code in ISO standard cards is 3 and immediately follows the Expiry code. For ANSI standard cards there is no equivalent to the SVC and a validation date(in yymm format) will immediately follow the expiry code. To properly assess this field it must be validated as either ISO or ANSI.

        For ISO validation:
        i. The length of the data area of the SVC is a minimum of 3 characters and

1. The 1st service code (SVC) digit is one of: **1,2,5,6,7,9**
2. The 2nd service code (SVC) digit is one of: **0,2,4**
3. The 3rd service code (SVC) digit is one of: **0,1,2,3,4,5,6,7**

Or,

For ANSI X4.16 validation:

i. The length of the data area is a minimum of 4 characters and
   1. The Validity Date contains only numeric data
   2. The numeric value of the 'mm' subfield is between 1 and 12

e. The first character is B

2. If Track 2 is present, then on track 2 all the following rules apply:

a. The length of the PAN is calculated by counting all non-space characters in the PAN field (ie, spaces are allowed, but excluded from the PAN length calculation)

   i. The length of PAN is greater than or equal to 12, and
   ii. The PAN area contains only numeral data and spaces, and
   iii. The PAN IIN prefix is not listed in the Platform Truncation Exclusion List (see below)

b. The field separator is '=', and

   i. The track contains exactly one separator

c. The length of the Expiry Code(in yymm format) is equal to 4, and

   i. The Expiry Date contains only numeric data
   ii. The numeric value of the 'mm' subfield is between 1 and 12

d. The length of the SVC code in ISO standard cards is 3 and immediately follows the Expiry code. For ANSI standard cards there is no equivalent to the SVC and a validation date(in yymm format) will immediately follow the expiry code. To properly assess this field it must be validated as either ISO or ANSI.

   For ISO validation:

   i. The length of the data area of the SVC is a minimum of 3 characters and
      1. The 1st service code (SVC) digit is one of: **1,2,5,6,7,9**
      2. The 2nd service code (SVC) digit is one of: **0,2,4**
      3. The 3rd service code (SVC) digit is one of: **0,1,2,3,4,5,6,7**

   Or,

   For ANSI validation:

ii. The length of the data area is a minimum of 4 characters and
   1. The Validity Date contains only numeric data
   2. The numeric value of the 'mm' subfield is between 1 and 12

3. If both Track 1 and Track 2 are present, then all the following rules apply in addition to the individual track rules for both tracks listed above:

a. The length of the PAN on Track 1 and the length of the PAN on Track 2 are equal (as calculated above, excluding spaces)
b. The data in the ISO Service Code or ANSI validity date area on Track 1 and Track 2 are equal.

4. If Track 1 is not present and Track 2 is not present, but the JIS-II track is present, then on the JIS-II track all the following rules apply:

a. The first character is alphabetical [A-Z, a-z]
b. The PAN IIN prefix is not listed in the Platform Truncation Exclusion List (see below)

To summarize: **If the track data that is present matches all of the rules that apply to those tracks, then the card is considered a payment card and truncation procedures listed in the next section apply to that card.**

- If a card is read with track 1 data but track 2 is missing or has no data, then only the rules in (1) apply
- If a card is read with track 2 data but track 1 is missing or has no data, then only the rules in (2) apply
- If a card is read and includes data on both track 1 and track 2, then all the rules in (1), (2) and (3) apply
- If a card is read but does not include track 1 or track 1 is empty, and does not include track 2 or track 2 is empty, but includes the JIS-II track, then all the rules in (4) apply

However, note the following:

- The determination of truncation for ISO/ANSI and. JIS-II is based on each individual set of rules for that track.
- If JIS-II is determined to be for credit, then it will be truncated regardless of the determination for ISO/ANSI tracks.
- Likewise, if the ISO/ANSI tracks are determined to be for credit, then they will be truncated regardless of the determination for the JIS-II track.

8.5 Data Truncation Flow Overview

Two modes of truncation, ie: FOID Data Record and DISCRETIONARY Data Record are defined below. This section specifically defines the data truncation rules that a CUSS platform must implement to be compliant with this chapter. All indexes are 1-based, as shown in the previous diagram, **and truncation ranges are inclusive of the start and end positions.** Examples of truncation are provided in the next section.

1. The platform must detect card track data that complies with the payment card standards listed above and apply the rules list above to determine if a card is a bank payment card data, or is not bank payment card data.

2. Any track data detected as being payment card data, and the data is subject to **FOID Data Record truncation**, then the platform must perform the following truncation (**note**: The PAN must be compacted by removal of all embedded spaces before applying the truncation rules below):

    a. If track 1 is present and not empty then truncate:

        i. All numeral data from position 8 to the 5th position before the first separator (mask PAN)
        ii. All data from the 1st position after the second separator until the end of the track (Expiry and SVC or Date)

    b. If track 2 is present and not empty then truncate:

        i. All numeral data from position 7 to the 5th position before the separator (mask PAN)
        ii. All data from the 1st position after the separator until the end of the track (Expiry and SVC or Date)

    c. If track 3 is present and contains at least one **=** separator, then truncate:

        i. All numeral data from position 7 to the 5th position before the first **=** separator (mask PAN)
        ii. All data from the 1st position after the first **=** separator until the end of the track (Security Code and Additional Data)

    d. If track JIS-II is present (defined as CUSS Track 4) and the 1st position is alpha, then truncate:

        i. All data from position 3 to position 6 (4 characters, PIN)
        ii. All numeral data from position 17 to position 22 (6 characters, mask PAN)
        iii. All data from position 40 to position 46 (7 characters, Expiry and Security Code)
        iv. All data from position 68 to the end of the track

3. Any track data detected as being payment card data, and the data is subject to **DISCRETIONARY Data Record truncation**, then the platform must perform the following truncation:

a. If track 1 is present and not empty then truncate:

   i. All numeral data from position 8 to the 1st position before the first separator (mask PAN)
   ii. Three data positions from the 5th position after the second separator (ISO SVC) or four data positions from the 5th position after the second separator (ANSI validation date).

b. If track 2 is present and not empty then truncate:

   i. All numeral data from position 7 to the 1st position before the first separator (mask PAN)
   ii. Three data positions from the 5th position after the second separator (ISO SVC) or four data positions from the 5th position after the second separator (ANSI validation date).

c. If track 3 is present and contains at least one **=** separator, then truncate:

   i. All numeral data from position 7 to the 1st position before the first **=** separator (mask PAN)

d. If track JIS-II is present (defined as CUSS Track 4) and the 1st position is alpha, then truncate:

   i. All data from position 3 to position 6 (4 characters, PIN)
   ii. All numeral data from position 17 to position 26 (10 characters PAN mask)
   iii. All data from position 44 to position 46 (3 characters, Security Code mask)

4. The truncation character must be **X** (ASCII character 0x58.) This character is a de facto payment industry standard for PAN masking and truncation.

In short, truncation replaces the PAN with the truncation character X, except for the leading six and trailing four digits. For more information on why this 6+4 truncation is used, please review the PCI-DSS FAQ site https://www.pcisecuritystandards.org/.

Payment card PAN values truncated with this 6+4 scheme are acceptable for purposes of identification, according to IATA RESOLUTION 722f with 2010 amended sections 20.60, 20.61, 20.62, 20.63, 20.64 and 20.66.

8.6 Data Truncation Exclusion List

The rules listed above are designed for the Platform to detect bank payment cards. It is possible that these rules will also detect some card as bank cards, even though those cards are not bank cards. This includes some card types that emulate the bank card track format, intentionally or not, but are not bank cards, or other unanticipated cases of false positive detection.

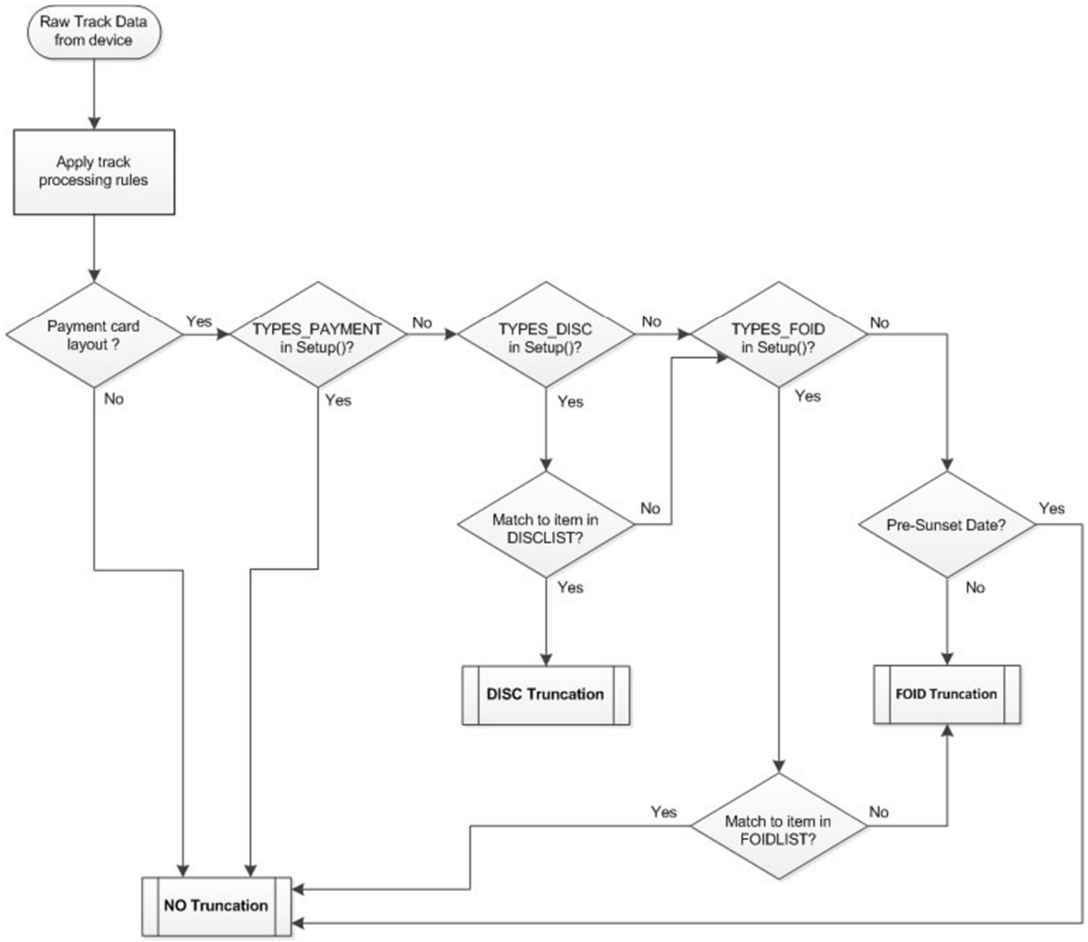
To avoid truncation for specific types of card, the platform will apply two **Data Truncation Exclusion Lists**: the FOIDLIST (do not apply any truncation) and the DISCLIST (apply Discretionary Data Truncation)

1. The FOIDLIST is optional and provided by the application during setup() as the bytestream parameter to the `DS_TYPES_FOID_ISO` and/or `DS_TYPES_FOID_JIS2` parameters.
2. The DISCLIST is optional and provided by the application during setup() as the bytestream parameter to the `DS_TYPES_DISCRETIONARY_ISO` and `DS_TYPES_DISCRETIONARY_JIS2` parameters.
3. The list must be provided each time the application calls setup() and will be in effect until the next setup() call, or the end of the application session.
4. The lists provided by the applications must be in ASCII text format: a sequence of 1 or more IIN (Issuer Identification Number) numbers separated by commas (0x2C).
5. Each IIN in the list must consist of four or more digits.
6. An IIN matches a PAN number if the complete IIN is an exact prefix of the PAN number.
7. If the platform reads a Payment Card and the application provided the DISCLIST, and the PAN matches any IIN in the DISCLIST, then **DISCRETIONARY Data Record Truncation** takes place, as described above.
8. If the platform reads a Payment Card and the application provided the FOIDLIST, and the PAN matches any IIN in the FOIDLIST, then **no truncation takes place.**
9. The DISCLIST takes priority over the FOIDLIST.
10. If the platform reads a Payment Card and the application has **not** setup() the `DS_TYPES_PAYMENT_ISO` or `DS_TYPES_PAYMENT_JIS2` types, then **FOID Data Record Truncation** takes place, as described above.
11. The platform may log the FOIDLIST and DISCLIST parameters provided by the application. This log can be used as a part of a regular audit or security review procedure, for example as input to a central exceptions list for forensics review, or other platform maintenance activities.

For example, to read cards as FOID cards, but exclude all truncation for cards starting with “1234” and “2345”, and return discretionary data for all cards starting with “34567” or “4567” or “9998”, then the application calls setup() with:

Record [0]: DS_TYPES_FOID_ISO bytestream [1234,2345]
Record [1]: DS_TYPES_DISCRETIONARY_ISO bytestream [34567,4567,9998]

8.7 Visual Representation of Truncation Rules



8.8 Examples of Data Truncation

Example Track Layouts for FFP and Credit Cards

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	97	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	102	104
FFP	Track 1	B	6	2	7	4	2	1	n	n	n	n	n	n	n	n	n	n	^	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	^	m	m	y	y	^	7	0	1	n	n	n	n							0	0	0	0	0	0	0	0	0	0	0	n	n
	Track 2	6	2	7	4	2	1	n	n	n	n	n	n	n	n	n	n	=	m	m	y	y	7	0	1	n	n	n	n	0	0	0	0	0	0	0	n	n
	Track 3	
	JIS-II	Not credit card: Service code not equal on track 1 and track 2
FFP	Track 1	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	^	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	^	m	m	y	y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Track 2	n	n	n	n	n	n	n	n	=	n	n	n	n																								
	Track 3	n	n	n	n	n	n	n	n
	JIS-II	Not credit card: PAN Track 1 length not eqaul PAN Track 2 length, Track 2 PAN length < 13
FFP	Track 1	n	n	n	n	n	n	n	n	^	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a																														
	Track 2	n	n	n	n	n	n	n	n	=	n	n	n	n																								
	Track 3	
	JIS-II	Not credit card: PAN length < 13
FFP	Track 1	B	n	n	n	n	n	n	n	n	n	n	0	0	0	0	0	0	^	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	^	y	y	m	m	0	0	0	0	0	0	a	a	n	n	n	n	n	n	n	n	n	n			a
	Track 2	n	n	n	n	n	n	n	n	n	n							=	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Track 3	
	JIS-II	Not credit card: Non-numeric data in PAN on Track 2, Invalid service code or date format Track 1 and Track 2
FFP	Track 1	B	n	n	n	n	n	n	n	n	n	^	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	^	m	m	y	y	^	a	^	9	8	8
	Track 2	n	n	n	n	n	n	n	n	n	0	m	m	y	y
	Track 3	
	JIS-II	Not credit card: PAN length < 13, Missing seperator in Track 2, Invalid service code or date format in Track 1 and Track 2
FFP	Track 1	B	2	1	1	7	n	n	n	n	n	n	n	n	n	n	^	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	^	y	y	m	m	7	9	9	y	y	m	m	n	n	n	n	n	n	n	n	n	n	n
	Track 2	2	1	1	7	n	n	n	n	n	n	n	n	n	n	=	y	y	m	m	7	9	9	y	y	m	m	n	n	n	n	n	n	n	n	n	n	n
	Track 3	9	5	2	1	1	7	n	n	n	n	n	n	n	n	n	n	=	y	y	m	m	n	n	n	n	n	n	n	n	n	n	n	=	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n
	JIS-II	Not credit card: Invalid service code or date digits
JIS-II Credit Card (Credit Data Record)	Track 1	B	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	^	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	^	m	m	y	y	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n			n	n	n						
	Track 2	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	=	m	m	y	y	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n
	Track 3	
	JIS-II	a	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	f	f	f	f	f	f	f	f	f	n	n	n	n	n	n	n	n	n	n	n	n	n	n
JIS Credit Card (FOID Truncation Data Record)	Track 1	B	n	n	n	n	n	n	X	X	X	X	X	X	n	n	n	n	^	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	^	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
	Track 2	n	n	n	n	n	n	X	X	X	X	X	X	n	n	n	n	=	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
	Track 3	
	JIS-II	a	n	X	X	X	X	n	n	n	n	n	n	n	n	n	n	X	X	X	X	X	X	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	X	X	X	X	X	X	X	f	f	f	f	f	f	f	f	f	n	n	n	n	n	n	n	n	n	n	n	n	X	X
JIS Credit Card (Discretionary Data Record)	Track 1	B	n	n	n	n	n	n	X	X	X	X	X	X	X	X	X	X	^	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	^	n	n	n	n	X	X	X	n	n	n	n	n	n	n	n	n	n	n	n	n			n	n	n						
	Track 2	n	n	n	n	n	n	X	X	X	X	X	X	X	X	X	X	=	n	n	n	n	X	X	X	n	n	n	n	n	n	n	n	n	n	n	n	n
	Track 3	
	JIS-II	a	n	X	X	X	X	n	n	n	n	n	n	n	n	n	n	X	X	X	X	X	X	X	X	X	X	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	X	X	X	f	f	f	f	f	f	f	f	f	n	n	n	n	n	n	n	n	n	n	n	n	X	X
Credit Card (Credit Data Record)	Track 1	B	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	^	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	^	y	y	m	m	y	y	m	m				n	n	n	n	n						f	f	f	f	f	f	f	f	f	f	f
	Track 2	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	=	y	y	m	m	y	y	m	m			Note: ANSI format
	Track 3	
	JIS-II	
Credit Card (FOID Truncation Data Record)	Track 1	B	n	n	n	n	n	n	X	X	X	X	X	n	n	n	n	^	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	^	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
	Track 2	n	n	n	n	n	n	X	X	X	X	X	n	n	n	n	=	X	X	X	X	X	X	X	X
	Track 3	
	JIS-II	
Credit Card (Discretionary Data Record)	Track 1	B	n	n	n	n	n	n	X	X	X	X	X	X	X	X	X	^	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	^	n	n	n	n	X	X	X	X				n	n	n	n	n						f	f	f	f	f	f	f	f	f	f	f
	Track 2	n	n	n	n	n	n	X	X	X	X	X	X	X	X	X	=	n	n	n	n	X	X	X	X
	Track 3	
	JIS-II	

8.9 Modifications to the CUSS Card Reader interface

The card reader interfaces defined in CUSS are modified using the approach described in Chapter 6. For more information on this approach, please read these sections:

Chapter 6: Extended Device & Media Type Handling
Appendix H: Extended Data Type List (DS_TYPES)

Four new extended data types are defined:

DS_TYPES_FOID_ISO	100	ISO track data *with* FOID Data truncation as defined above
DS_TYPES_FOID_JIS2	14100	JIS-II track data *with* FOID Data truncation as defined above
DS_TYPES_PAYMENT_ISO	200	ISO track data *without* truncation
DS_TYPES_PAYMENT_JIS2	14200	JIS-II track data *without* truncation
DS_TYPES_DISCRETIONARY_ISO	300	ISO track data *with* discretionary data truncation as defined above
DS_TYPES_DISCRETIONARY_JIS2	14300	JIS-II track data *with* discretionary data truncation as defined above

These types replace types DS_TYPES_ISO and DS_TYPES_JIS2, which are now deprecated for card reader devices in CUSS Technical Specification 1.3. CUSS platforms must support these new data types in accordance with the existing Extended Media Type support methodology:

1. Other than modifications for Extended Media Type support, the behaviour of the card reader components and characteristics remain as they are currently defined elsewhere in this Technical Specification.

2. Advertise support for the extended DS_TYPES by including the correct settings within the component characteristics.

3. Allow the application to select which data types it receives by use of the setup() command with the correct parameters. If the application requires a truncation exclusion list for the particular data type, it must include that list as the bytestream field of the data record provided to the setup() command as per Section 6.2.1. See the Data Truncation Exclusion List section above for more details.

    a. The optional bytestream parameter for the `DS_TYPES_DISCRETIONARY` records is the “DISCLIST”

    b. The optional bytestream parameter for the `DS_TYPES_FOID` records is the "FOIDLIST"

4. Apply the truncation required to the track data if the card is a payment card and the requested data type is DS_TYPES_FOID_ISO, DS_TYPES_FOID_JIS2, DS_TYPES_DISCRETIONARY_ISO, or DS_TYPES_DISCRETIONARY_JIS2.

5. If the application does not call setup(), the platform returns the **default media type** (see next section) for the card reader, but the data type included in the msgDataType structure must be DS_OK (0) for backwards compatibility.

6. In accordance with Section 6.2.1, once the application calls setup(), those settings are in effect for the card reader until the application subsequently makes an additional setup() call, the application calls disable(), or the application transaction ends, whichever comes first.

7. The CUSS platform must implement this extended data type support and component model in accordance with Section 6.3.4.

8.10 Backwards Compatibility of Platforms and Applications

A goal of this change to the CUSS Technical Specification is to allow CUSS applications to continue to operate without requiring modification, when running on a CUSS platform that has implemented these CUSS 1.3 card reader behavior changes.

In other words, the change is transparent to existing applications, except the data they receive for payment cards is truncated. (Some CUSS applications do not perform payments and read cards only for FOID transactions.)

The changes in this Chapter must be implemented so that the platform remains backwards compatible with these applications, and so that these CUSS applications do not require any interface modifications in order to continue to receive FOID data (albeit the data is now truncated when bank card standards are detected).

So an updated CUSS platform must ensure that:

1. The platform must implement a **default media type** to use for its card reader components.

    a. The default media types must be DS_TYPES_FOID_ISO and DS_TYPES_FOID_JIS2, the truncated forms

2. If the applications uses the card reader MediaInput component but the application does not call setup() prior to enabling the device, then when the platform reads a card:

    a. The data returned to the application must be of the default media type as defined above

    b. The DS indicator within the msgDataType structure for card read events must be DS_OK (0), for compatibility -- since legacy applications would likely treat non-zero DS values as errors

3. If the CUSS platform implements the Extended Media Type support as multiple MediaInput components, then:

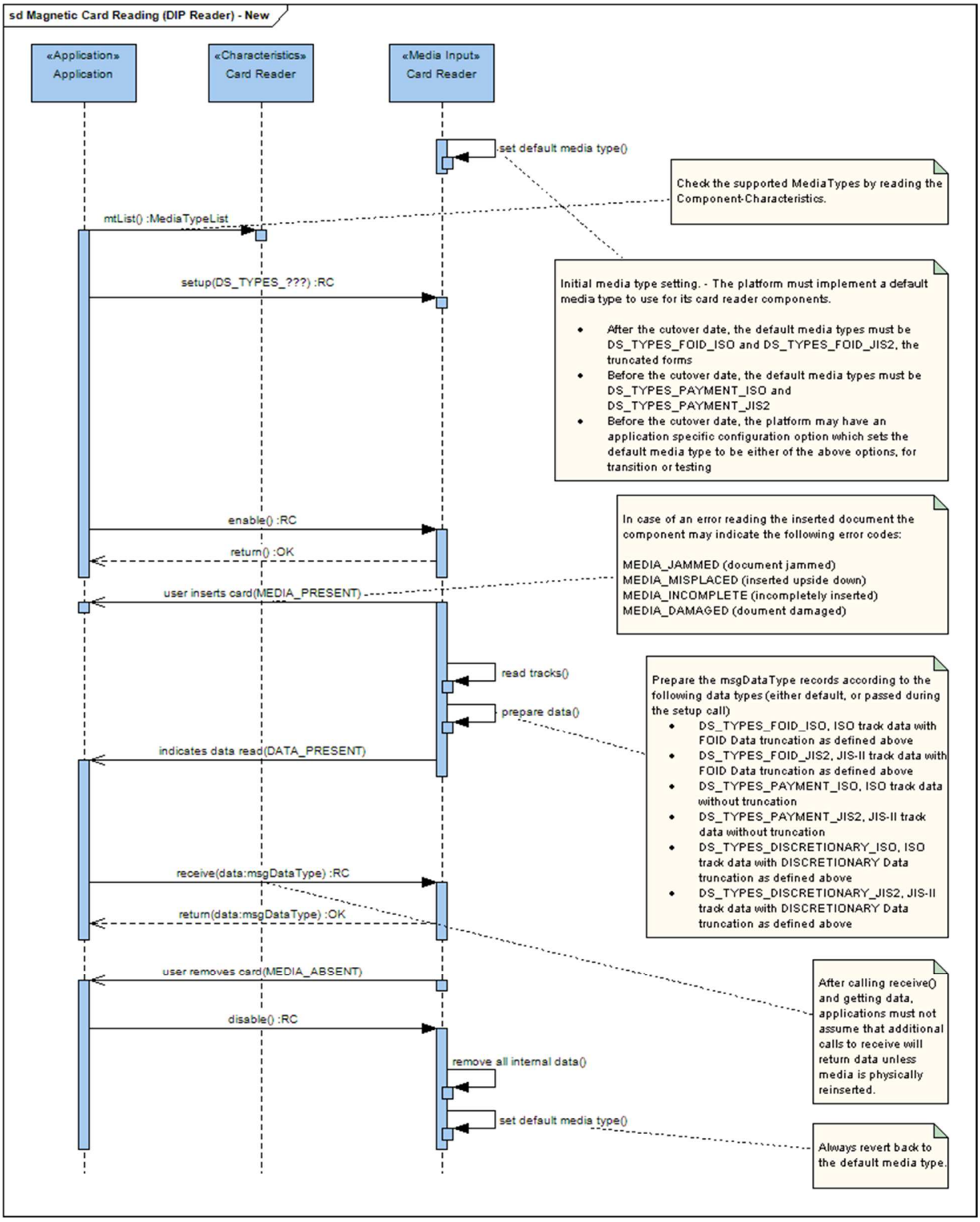
    a. An application that is following the card reader guidelines of Chapter 7 must find and detect the MediaInput component that is assigned the **default media type** as defined above

    b. If the application does not call setup(), the DS indicator within the msgDataType structure for card read events must be `DS_TYPES_ISO`, for compatibility (since non-zero DS values will likely be treated as errors)

    c. If an application also acquires the alternate MediaInput component, then the behaviour for component is as normally defined in the existing Extended Media Type support methodology

This approach ensures that applications that are not modified can continue to use card reader components as they do now, except that after a certain date (or when a configuration flag is set) the track data it receives from the platform is truncated.

8.11 Use Cases and Device Sequence



8.12 Deferred use of Payment Card Data

This section provides background and clarity on a CUSS application use case called "use of deferred payment data". It does not modify the Technical Specification.

Example Use Case/Business Requirement:

> *The passenger swipes a magnetic card during the booking identification phase. The card they read was a payment card, and they were successfully identified.*
>
> *Some later point in the transaction requires payment. The application would then re-use the same payment card data provided during identification, to complete the payment, instead of prompting the passenger to re-read the same card.*
>
> *The goal is to provide a seamless and logical overall transaction without requiring duplicate card reads.*

If the application wishes to have a "use of deferred payment data" use case as part of its business logic, it must request and receive the full payment data at the beginning (not just the FOID data) and preserve it within its logic until it needs it at a later point. The platform must not do this on the application's behalf.

If the application chooses to do this, it must read full track data during the identification phase, and protect that data for the entire time it is storing it in memory. This means that all phases of the application are in scope for PCI-DSS, not just the payment phase. Hence the decision to implement this transaction flow is a **business logic and security decision,** not a technical limitation.

Here is a reminder of existing CUSS Technical Specification requirements related to reading data from the kiosk platform.

1. **Section 6.2.1:** The application must call setup() prior to calling enable(), to indicate which types of extended data it wants. To request a different type of data, the application needs to call disable(), and then setup() with new parameters indicating the new data type. This means a card must be physically read from the user if a different type of data is needed.

2. **Section 6.3.1:** The application must call the receive() directive to access the data. The platform will provide the data types requested by the application via the most recent setup() call for that component.

3. **Section 3.6.8.1 Note 4:** The platform must erase its internal call data as soon as the application calls receive(), disable(), or the application ends its transaction, whichever comes first. Subsequent calls to receive() shall not return any data; hence the payment data cannot be cached for later use.

8.13 Deployment Guidelines and Instructions

The changes described in this platform are mandatory for all CUSS sites to remain compliant with IATA Recommended Practice RP1706C (effective date June 30th 2012.) This date strictly imposes these conditions:

1. Any kiosk deployed after this date which does not implement either the changes in the CUSS FOID Addendum document, or a fully compliant CUSS 1.3 platform, is not a CUSS-compliant kiosk.

2. Sites that do not deploy CUSS 1.3 must work with their kiosk suppliers to retroactively change a all existing deployments of CUSS 1.0, CUSS 1.1, and CUSS 1.2 to support the CUSS FOID Addendum.

3. Existing and new CUSS sites must work with their platform suppliers to ensure appropriate platform updates are deployed, if they wish to maintain site compliance with the CUSS Technical Specification.

4. If a CUSS application is running on a CUSS 1.3 platform and the application does not implement the changes in this Chapter, that application will not have access to full payment card track data: they will only receive truncated data.

5. Any CUSS application that needs full payment card track data for payment transactions must be modified to follow the changes in this Chapter. An application which does not do this will lose the ability to process payments within their kiosk application.

6. According to the *Appendix I: Application Updates and Distribution*, if an application is changed to use the new CUSS card reader extended media types, including new calls to setup(), then this would be a Level 1 Change. This type of change could be subject to testing or certification prior to deployment. Platform and application providers should discuss this on a case by case basis, to determine what level of testing is appropriate.

7. By examining the card reader characteristics, a CUSS application can determine if the platform us operating CUSS 1.3 or is implementing the CUSS FOID Addendum on previous versions of the CUSS Technical Specification. **It is an application business logic decision whether or not to operate in this environment after the cutover date.**

CUSS sites may choose to work with their platform supplier and application providers to coordinate transition period to CUSS 1.3 in preparation for this change.

Platform providers control all access to kiosk device components and interfaces on their CUSS platform. Once this change is implemented, additional control is available to the platform provider in granting or denying CUSS application access to card reader data (because there are now different requests for different types of data.)

At their discretion, platform providers may choose to leverage this control and add additional business rules that could be used to restrict CUSS application access to payment card data.

For example, a platform could only permit access to payment card data to known or specific tested versions of a CUSS application. A platform provider shall discuss any such restrictions with the affected application providers, prior to enacting them.

Summary of Platform Changes (mandatory for CUSS 1.3):

1. Implement new components and characteristics to implement the new extended media types.

2. Detect when a payment card is read as part of a CUSS application transaction

3. Apply the truncation rules to modify the raw track data if needed and in accordance to the component characteristics, the type of card read, and the extended media type requests made by the application

4. Ensure that all implementation is done so that legacy application maintain access to the card reader components and data (but will only receive truncated data)

Summary of Application Changes (mandatory only if access to payment data is required):

1. If the application does not require full PAN data for any aspect of its business logic, no interface changes are needed to receive FOID data from the CUSS reader.

2. Even if the application does not use full payment card data, verify that internal card track parsing logic can tolerate truncated data without error. For example, a FOID transaction

might fail if an application expects all characters in the PAN area of a payment card to be digits, and it instead receives data truncated with ‘X’, even if ultimately it only uses the name data.

3. Applications that require full payment track data need to change so that:\

    a. They can detect and enable the specific card reader MediaInput component that includes support for the new extended `DS_TYPES_PAYMENT` types.

    b. The correct data type is selected using setup() at the right points in the transaction; ie, select the `DS_TYPES_PAYMENT` component and call setup() for that type, for payment transactions, and select the `DS_TYPES_FOID` and/or `DS_TYPES_DISCRETIONARY` component and call setup for that type, for FOID and all other non-payment card reading transactions.

4. If the application uses the PAN number as a search criteria, verify if other aspects of the application architecture require change (such as DCS search methods) to support 6+4 truncated data

Ch 9: Automated Remote Updates (ARU)

This chapter defines the requirements for automated remote updates (ARU) for CUSS platforms and applications. It is a combination of business requirements as well as Technical Specification changes in CUSS 1.3.

Note: An automated remote updates process is meant to be an option available to airlines that wish to use it. Airlines are not forced to use ARU.

Background

Automated Remote Updates (ARU) has been in use in the CUSS environment for years. However, there is no standard for how to accomplish ARU and no requirement that all Platforms must facilitate ARU.

There are benefits to Airlines, Platform Operators, and Platform Suppliers in making ARU a standard part of CUSS products. These benefits include:

- Reduces costs and efforts otherwise required for the Platform Suppliers to receive and manage multiple files for update requests arriving from the entire community of Airlines operating at all of their supported sites.
- Reduces costs and efforts otherwise required from each Platform Supplier to prepare, test and document a final installation package for each release to all of their sites.
- Reduces the risks of omissions or errors (and re-works) in the resulting Platform installation package.
- Reduces the coordination efforts otherwise required between Airlines, Platform Suppliers and each individual CUSS site Operator to execute and validate a “completed installation”.
- Reduces the level of activity by local technicians at the individual CUSS sites in the installation and management of Airline Application releases.
- Reduces the risks, errors and re-work often encountered when the local site does not precisely comply with the exact installation instructions.
- Reduces the tedious/unproductive efforts invested by all parties for cases of faulty installation.

- Mitigates risks by supporting an initial Beta rollout to selected sets, provides an automated rollback when required and a global deployment capability without committing multiple resources from all parties.

- Provides forward-looking technologies that leverage the CUSS standards for optimal capability and functionality.

Goals

This section defines business requirements for ARU. Later in the chapter, the specific technical features of the CUSS standard allowing ARU are described. Generally speaking, an ARU process:

- Must ensure consistency of the deployment process.
- Must ensure that applications are installed correctly.
- Must reduce the amount of time that it takes to deploy eligible application updates.
- Must provide verifiable results of the deployment
- Requires that Application Providers trust the checks and balances put in place by the specifications to prevent adverse impact on the common use environment.
- Requires that Airport operations must not be impacted.

Business Requirements

General

1. The processes of the ARU tool must facilitate Deployment to specific workstations and/or kiosks.

Certification

1. Only specific types of application updates are eligible to be deployed using the ARU process.
    a. Changes invalidating the PCI compliance of an application, platform, or airport are not eligible for deployment using the ARU process.

2. Application updates deployed through ARU must complete a Beta Test successfully prior to Global Release.
    a. This Beta Test must be executed using the ARU process.

3. An "ARU Tool" may be provided by the Application Supplier, Application Provider, Platform Supplier, Platform Operator, or other party. An airline is not required to implement such a tool but may opt to obtain one from another party.

4. An ARU Tool must be certified before use in a production environment.
    a. An ARU tool is subject to the same certification requirements as the common use application it is updating.

Notification (Order/Change Request)

1. For each instance of the ARU process, the Stakeholders must be identified. The Stakeholders include defined representative(s) of the following. Note that the members of the Stakeholders do not necessarily have to act on the notification.
    a. The airline requesting the ARU
    b. The airlines operating at the airport
        i. The idea here is that any change, no matter how small, may have side effects. Notification of changes is key to the quick determination of any negative impacts of a change.
        ii. These other airlines could become part of the 'fyi list'.
        iii. If the other airlines do not really care, then the site admin/support personnel can handle this determination, and this portion of the stakeholders can simply be deleted.
        iv. Example: 10 airlines are running on a CUSS kiosk. Airline1 makes an update with ARU. One or more of the other 9 airlines on that kiosk begin to have problems. For the other 9 airlines, knowledge of the update by Airline1 may help to determine negative impacts by Airline1's update.
    c. The airport, such as its Change Control Board, managers, site administrators, or other personnel.
    d. Platform Supplier
    e. Platform Operator
    f. Any organization responsible for the Service Level Agreement of the airport, airline, application, or platform.

2. The ARU Stakeholders must define acceptable change control windows for each location.

3. The Application Provider/Supplier must notify the Stakeholders of their intent to deploy an application update using ARU.

4. At the time of notification the Application Provider/Supplier must provide to the Platform Supplier the complete application package including all updates to be made.

5. The Stakeholders must review the notification to identify any conflicting deployments previously scheduled. A Stakeholder may request that the Application

Provider/Supplier reschedule a proposed ARU based on previously-scheduled deployments.

Parameters for Distribution and Activation

1. The application update must include information that allows the Platform to validate that the carrier is permitted to utilize the ARU process.
    a. The Platform must be permitted to decline application updates from Application Provider/Suppliers whose ARU process has previously adversely affected the common use environment.

2. The Platform Operator must be able to decline temporarily all ARU attempts from any Application Provider/Supplier.
    a. The purpose of this function is to facilitate short lived site specific problem resolution or site freezes. If a an ARU is temporarily declined the reason for this needs to be communicated to the requesting airline.

3. The application update must include information that allows the Platform to validate that the update is eligible to be deployed and/or activated.

4. The Platform must provide an interface to Application Providers/Suppliers to communicate information regarding the common use environment at the target site.

5. The application must include easily-accessible information to the platform, indicating the version that is in use. Every change requires the application to change its version.

6. The distribution of application updates must not exceed a defined percentage of the available bandwidth capacities or protocol limits. This limit is to be set by the Platform Operator.

7. The distribution and activation of an application update must not cause any workstation or kiosk to exceed the maximum CPU utilization.

8. The Platform must perform a real-time virus check on the received files included in the application update at the time of Distribution.

9. The distribution and activation of an application update must not render the workstation or kiosk unavailable for use by other applications.
    a. This is a requirement for both the Application Provider/Supplier and the Platform Supplier.

10. The Platform must allow the ARU tool to re-start the Application without rebooting the workstation or kiosk.

Distribution

The ARU process must allow an Application Provider/Supplier to distribute application updates subject to eligibility rules defined in Appendix I to target locations independent of intervention from the Platform Supplier or Platform Operator.

Activation

1. The ARU process must allow an Application Provider/Supplier to activate an application update to target locations independent of intervention from the Platform Supplier or Platform Operator, subject to eligibility rules defined in Appendix I.

2. Activation will not proceed without disaster recovery being available. Refer to fallback/rollback for clarification

Validation

1. The ARU Tool must check whether the activation was successful or not.
    a. The ARU tool cannot provide application functional validation. This can only be done by end users.

2. The ARU tool should provide capability to indicate the need for a rollback/fallback in case of unsuccessful activation.

Cleanup

1. The ARU tool must adhere to the clean up rules defined in the CUPPS TS and CUSS TS.

Rollback/Fallback

1. Prior to Activation the Platform Supplier/Operator must ensure to preserve the current version of the application.

2. The Application Provider/Supplier must provide Rollback documentation

3. The ARU Tool must have the capability to rollback the application anytime if required.
    a. Ensure proper notification prior to Rollback.

4. The Rollback execution will be executed by the ARU Tool or by the Platform Supplier/Operator as defined in the Rollback plan.

5. The Fallback execution shall remain the sole responsibility of the Platform Suppliers/Operator.
    a. The Platform Supplier/Operator can temporarily disable ARU to ensure that the Rollback/Fallback returns the Platform to its previous state.
    b. The Platform Supplier/Operator will engage with the Airline to resolve the issue(s).

Service Level Agreement

1. The SLA should take into account adverse impact caused by an Application Provider/Supplier rather than the Platform Supplier or Platform Operator.

2. Airlines using ARU must have a 24/7 major incident management process and resources.

Application ARU via the CUSS Technical Interfaces

Applications that wish to perform automated remote updates on a CUSS kiosk must comply with the business requirements described above, and use the new CUSS interface methods added to CUSS 1.3 to request permission to perform an ARU.

In prior versions of CUSS, applications could perform remote updates, without control or oversite by the platform. This is an implicit capability of the CUSS standard, which allows applications completely control over their business logic and local storage directory, and allows Application Service Provider interfaces to stop and restart an application.

If the platform does not support CUSS 1.3, the application may defer to previous update methods implicitely supported by the CUSS standard.

If the platform does support CUSS 1.3, the application must follow the ARU guidelines outlined here. The application ARU process shall be:

1. Generally speaking, the ARU infrastructure of the application must comply with the business guidelines listed above. These are non-technical requirements that cannot be prescribed within the interface definition of the CUSS Technical Standard.

2. The technical requirements listed here can be done from within the CUSS application itself, or via an Application Service Provider application connected to the CUSS platform System Manager interface.

3. At startup, the application must perform the **VERSION_EXPLANATION** request described in Section 2.4.5.8, if it intends to perform automated remote updates.

4. In response to the **VERSION_EXPLANATION**, the platform will indicate the parameters affecting ARU for that application:
    a. Time of day restrictions for applying updates
    b. Download bandwidth restrictions
    c. Suggested CPU limit during update

5. While operating normally, the application can perform tasks related to ARU (such as background downloading) within the guidelines of the paramers provided as per 2.4.5.8. In particular:

    a. Background transfers to download updates can take place at any time and are subject to the bandwidth limitation provided for ARU.
    b. The updates can only be installed during the time window restriction provided for ARU.

6. When the application is ready to perform the update, it must send an **UPDATE_REQUEST** event to the platform as described in Section 2.4.5.9

7. The platform can evaluate the name of the application making the request, the version it reported at startup via the VERSION_EXPLANATION request, and the version it is requested to upgrade to via the UPDATE_REQUEST.

8. This information, along with internal provider back office procedures, are sufficient to evaluate if the application is permitted to update on this particular kiosk at this time. If the platform then returns RC_OK, the application is permitted to perform the ARU. If the platform does not return RC_OK, the update is not permitted.

9. If the platform responds RC_PARAMETER to either the **VERSION_EXPLANATION** or **UPDATE_REQUEST** events, the application can assume the platform does not support CUSS 1.3 ARU interfaces, and can revert to previous update behaviours.

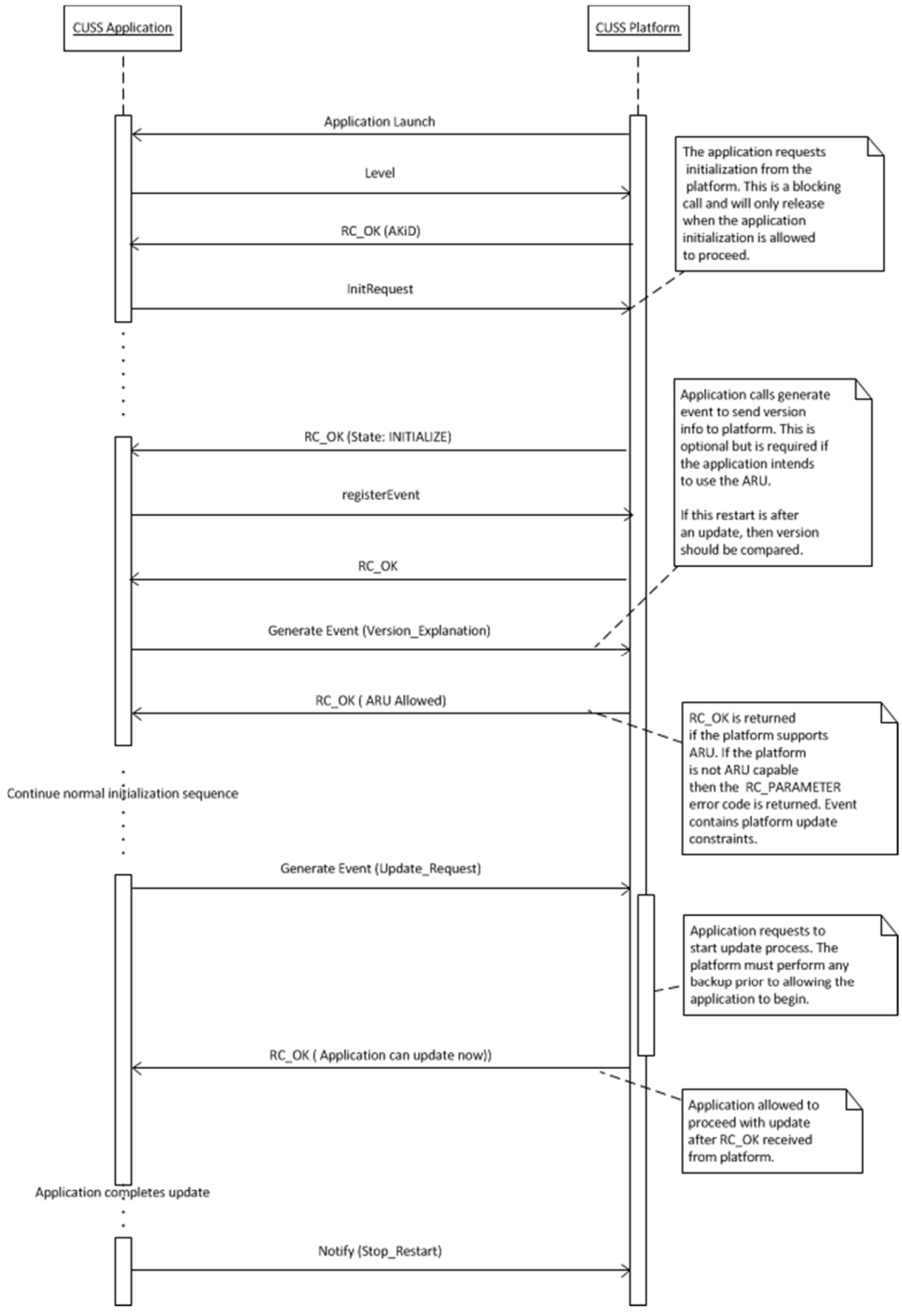
10. Either immediately, or within the time window specified by the platform to allow ARU, the application can then take the technical steps needed to apply the updates to the local application files.

11. Once the technical update steps are complete (this will vary from application to application) the application can request an application restart, by using the notify() request using state transition AVAILABLE_STOPPED_RESTART, UNAVAILABLE_STOPPED_RESTART, or ACTIVE_STOPPED_RESTART.

12. Once restarted, the application must again report its version using the **VERSION_EXPLANATION** event.

    a. If the application chose to not apply the ARU or there was an error applying the update, this version report should be the same as the previous restart.
    b. If the ARU was applied successfully, this version report should match the version requested as part of the UPDATE_REQUEST.

Here is a sequence diagram of a typical normal update that is successfully performed using the ARU interfaces:



Appx A: Return, Event and Status Codes

This Appendix lists all functions return codes, event codes, status codes used in CUSS.

Function Return Codes

Function return codes describe the lexico-syntaxic analysis results of an interface call (directive). If the return code is 0 (RC_OK), then the directive has been accepted. Negative return codes means that the directive has been rejected.

Function Return Codes		
Code	**Name**	**Description**
0	RC_OK	Directive accepted. No errors detected by function.
-1	RC_REFERENCE	Invalid application reference. The calling application is using an invalid application token that has not been assigned by application manager..
-2	RC_STATE	Invalid state. The application is not in the correct state to invoke the called function.
-3	RC_DENIED	Access denied. The application is not allowed to use the function.
-4	RC_PARAMETER	Parameter error. An error has been detected in the passed arguments to the called function.
-5	RC_ANY_PARAMETER	Error in using a CORBA::any type. The data type contained in a CORBA::any type maybe unusable. e.g. The datastream parameter, an alias of CORBA::any, passed in the send/setup functions is not one of he accepted data types; aeaDataType, svgDataType, msgDataType or nilDataType.
-6	RC_LISTENER	No listener set. No listener reference has been set for asynchronous events.
-7	RC_SHARE	Invalid share mode. This is used when SP system manager is trying to have exclusive access of a device when it is not allowed.
-8	RC_UNAUTHORIZED	Attempted to issue an unauthorized command. e.g. The DataStream parameter passed in the send/setup function may contain a non-accepted ATB or SVG command and/or data.
-9	RC_ERROR	Any other error not defined above
-10	RC_NOT_SUPPORTED	This function is not supported (i.e. not implemented).

Event Codes[45]

An event code reflects either an application or component state transition (or the actual state itself in case there is no state transition). For more detailed description of application states and state transitions, refer to Sections 2.4.1 and 2.4.3. For more detailed descriptions of device component states and state transitions, refer to Sections 2.6.3 and 2.6.5.

Event Codes			
Code	**Name**	**Usage**	**Description**
000	EC_OK	System Manager	Used in the returned event for calls to **`suspendAll`**, **`resumeAll`** or **`stopAll`** directives.
Component State Transitions			
001	EVENTHANDLING_ READY	Peripheral	The application has successfully executed a directive for a device component that is online and functioning normally (soft conditions and OK only). This is also used when a device component recovers from a hard condition.
002	UNAVAILABLE_ RELEASED_ PLATFORM	Peripheral	An authorized platform component has released a device component that is offline or not functioning normally (e.g. before CAM moves an application to DISABLED state)
003	EVENTHANDLING_ UNAVAILABLE	Peripheral	An acquired device component has now become unavailable due to a hard condition such as becoming offline or unusable.
004	UNAVAILABLE_ RELEASED_ APPLICATION	Peripheral	The application has released a device component that is offline or not functioning normally.
005	READY_RELEASED_ APPLICATION	Peripheral	The application has released a device component that is online and functioning normally.
006	READY_RELEASED_ PLATFORM	Peripheral	An authorized platform component has released a device component that is online and functioning normally (e.g. before CAM moves an application to DISABLED state)
007	RELEASED_READY	Peripheral	Application has successfully acquired a device component that is online and functioning normally.
008	RELEASED_UNAVAILABL E	Peripheral	Application has successfully acquired a device component that is offline or not functioning normally (unusable).
Application State Transitions			

[45] In CUSS 1.0, event codes 117, 124, 125, 126, and 131 are no longer used.

Event Codes			
Code	**Name**	**Usage**	**Description**
101	INITIALIZE_DISABLED	CAM -> App	State transition **Disable**: CAM moves an application into DISABLED state due to incorrect behavior while in INITIALIZE state.
102	AVAILABLE_ DISABLED	CAM -> App	State transition **Disable**: CAM moves an application into DISABLED state due to incorrect behavior while in AVAILABLE state.
103	ACTIVE_DISABLED	CAM -> App	State transition **Disable**: CAM moves an application into DISABLED state due to incorrect behavior (e.g. KILL_TIMEOUT expires) while in ACTIVE state.
104	UNAVAILABLE_ AVAILABLE	App -> CAM	State transition **Wait**: Application has requested to move to AVAILABLE state after determining that the CUSS environment is adequate to its proper execution.
105	AVAILABLE_ACTIVE	CAM -> App	State transition **Activate**: CAM moves an application into ACTIVE state after user has selected its icon on CLA
106	ACTIVE_AVAILABLE	App -> CAM	State transition **Wait**: Application has requested to move back to AVAILABLE state after completing its session.
107	INITIALIZE_STOPPED_ST OP	App -> CAM CAM -> App	State transition **Stop:** Application or SM has requested to stop the application while it is in INITIALIZE state.
108	AVAILABLE_STOPPED_S TOP	App -> CAM CAM -> App	State transition **Stop:** Application or SM has requested to stop the application while it is in AVAILABLE state.
109	ACTIVE_STOPPED_ STOP	App -> CAM CAM -> App	State transition **Stop:** Application or SM has requested to stop the application while it is in ACTIVE state.
110	SUSPENDED_ STOPPED_STOP	CAM -> App	State transition **Stop:** CAM moves an application into STOPPED state upon request from the same SM that had suspended it.
111	DISABLED_STOPPED_ST OP	CAM -> App	State transition **Stop**: CAM or SP SM has requested to stop an application while it is in DISABLED state after human intervention has occurred.
112	SUSPENDED_ AVAILABLE	CAM -> App	State transition **Resume:** CAM moves an application back to AVAILABLE state upon request from the same SM that had suspended it.
113	AVAILABLE_ SUSPENDED	CAM -> App	State transition **Suspend**: CAM moves an application from

Event Codes			
Code	**Name**	**Usage**	**Description**
			AVAILABLE state to SUSPENDED upon request from a SM.
114	INITIALIZE_STOPPED_RESTART	CAM -> App	State transition **Restart:** Application will be stopped and reloaded by CAM due to system restart.
115	AVAILABLE_ STOPPED_RESTART	CAM -> App	State transition **Restart:** Application will be stopped and reloaded by CAM due to system restart.
116	ACTIVE_STOPPED_ RESTART	CAM -> App	State transition **Restart:** Application will be stopped and reloaded by CAM due to system restart.
118	SUSPENDED_ STOPPED_RESTART	CAM -> App	State transition **Restart:** Application will be stopped and reloaded by CAM due to system restart.
119	STOPPED_INITIALIZE	CAM -> App	State transition **Load:** CAM loads a STOPPED application upon request from SM or from itself.
120	DISABLED_INITIALIZE	CAM -> App	State transition **Load**: CAM loads a DISABLED application upon request from SM or from itself after human intervention occurs.
121	UNAVAILABLE_ STOPPED_RESTART	CAM -> App	State transition **Restart:** Application will be stopped and reloaded by CAM due to system restart.
122	UNAVAILABLE_ DISABLED	CAM -> App	State transition **Disable**: CAM moves an application into DISABLED state due to incorrect behavior while in UNAVAILABLE state.
123	UNAVAILABLE_ SUSPENDED	CAM -> App	State transition **Suspend**: CAM moves an application from UNAVAILABLE state to SUSPENDED upon request from a SM.
127	SUSPENDED_ UNAVAILABLE	CAM -> App	State transition **Resume**: CAM moves an application back to UNAVAILABLE state upon request from the same SM that had suspended it.
128	UNAVAILABLE_ STOPPED_STOP	CAM -> App	State transition **Stop**: Application or SM has requested to stop the application while it is in UNAVAILABLE state.
129	INITIALIZE_ UNAVAILABLE	App -> CAM	State transition **Check**: Application has requested to move to UNAVAILABLE state after completing its initialization.
130	AVAILABLE_ UNAVAILABLE	App -> CAM	State transition **Check**. Application has requested to move back to UNAVAILABLE state after determining that the CUSS environment is not adequate to its proper execution.

Event Codes			
Code	**Name**	**Usage**	**Description**
132	ACTIVE_ACTIVE	App -> CAM	State transition **Wait**: Application has indicates a customer transaction has started while in Persistent Single-Application Mode.
133	ACTIVE_UNAVAILABLE	App -> CAM	State transition **Check**. Application has requested to move back to UNAVAILABLE state after determining that the CUSS environment is not adequate to its proper execution.
Application/Component States			
201	RELEASED	Peripheral	A device component has been released (or not yet acquired) by the application. It may or may not be usable.
202	UNAVAILABLE	Peripheral	A previously acquired device component is offline or not functioning normally (unusable).
		Application	Application is in UNAVAILABLE state .
203	READY	Peripheral	A previously acquired device component is online and offline or not functioning normally (ready to be used).
204	STOPPED	Application	Application is in STOPPED state.
205	SUSPENDED	Application	Application is in SUSPENDED state.
206	DISABLED	Application	Application is in DISABLED state.
207	INITIALIZE	Application	Application is in INITIALIZE state.
208	AVAILABLE	Application	Application is in AVAILABLE state.
209	ACTIVE	Application	Application is in ACTIVE state. CAM may use this event code to ask the application to complete its session if SESSION_TIMEOUT has elapsed.
210	BUSY	Peripheral	A device component is in transient state BUSY (e.g. reading/writing under progress)

Status Codes

Status code describes the current status of a component or the result of the semantical analysis of a component interface call or the execution result of a component interface call. Refer to the status code tables for all the device components directives in Section 3.6 to find in which context these status codes must be used.

Status Code Description			
Code	**Name**	**Description**	**Event Type**
000	OK	Device online and ready or interface calls generates no error.	public[1]
001	TIMEOUT	Synchronous or asynchronous function call timed out	private
002	WRONG_STATE	Component is in the wrong state to receive this call	private, platform
003	CANCELLED	Asynchronous function call cancelled.	private
004	SOFTWARE_ERROR	Detected a recoverable software error during execution of function.	private, platform
005	ALMOST_OUT_OF_ TIME	NOT used in CUSS1.0	private, platform
006	OUT_OF_SEQUENCE	Function has been called out of sequence. (e.g. calling send/receive before enable or calling enable/disable twice)	private
Media-Related (100-199)			
101	MEDIA_JAMMED	Documents or magstripe card jammed inside device.	public
102	MEDIA_MISPLACED	Document or magstripe card inserted incorrectly. e.g. An ATB document inserted up-side down.	private, platform[2]
103	MEDIA_PRESENT	Document is inserted into device. A event must be sent even if media was not kept in the peripheral device (e.g. swipe or DIP card reader)	private
104	MEDIA_ABSENT	No document to offer (for Dispenser/Feeder) or document is removed from device while it is enabled (for MediaInput, MediaOutput). In the latter case, an event must be sent even if document was not kept in the peripheral device (swipe or DIP card reader)	private
105	MEDIA_HIGH	Component (e.g. Capture) reached AlmostFullLevel Threshold	public
106	MEDIA_FULL	Feeder/Dispenser/Capture device is full of documents.	public
107	MEDIA_LOW	Feeder reached AlmostEmptyLevel Threshold.	public
108	MEDIA_EMPTY	Feeder has no documents (e.g. out of paper)	public
109	MEDIA_DAMAGED	e.g. Card/Coupon physically damaged	public
110	MEDIA_ INCOMPLETELY_ INSERTED	Document is incompletely inserted into device and removed	private

[1] private for solicited events
[2] only for userless classes

Status Code Description			
Code	**Name**	**Description**	**Event Type**
Data-Related (200-299)			
201	FORMAT_ERROR	Error detected in format of data used in send/receive. e.g. An ATB card/coupon incorrectly encoded or wrong PECTAB or invalid datastream (AEA99 or SVG).	private, platform[3]
202	LENGTH_ERROR	Data stream provided for send/receive is incomplete.	private, platform[3]
203	DATA_MISSING	No data provided while using send/receive functions.	private, platform[3]
204	PHYSICAL_ERROR	Not used in CUSS 1.0	private, platform[3]
205	DATA_PRESENT	DATA read from the inserted document and application can get this data by calling receive.	private
Hard Error Related (300-399)			
301	CONSUMABLES	e.g. printer ribbon , head	public
302	HARDWARE_ERROR	An error due to hardware malfunction that makes the component UNAVAILABLE	public
303	CRITICAL_ SOFWARE_ERROR	Detected a software error during execution of function that makes the component UNAVAILABLE	public
304	NOT_REACHABLE	Device is not connected (unknown status) .	public
305	NOT_RESPONDING	Device is connected but not responding or not ready	public
306	THRESHOLD_ERROR	Too many errors have occurred.	public
307	THRESHOLD_USAGE	Inserting/removing card/coupons performed too many times.	public
308	CONFIGURATION_ ERROR		public
309	SESSION_TIMEOUT	Active Application Session timeout. Sent when sessionTimeout elapses.	private, platform
310	KILL_TIMEOUT	Sent before moving an application to DISABLED state.. Sent after killTimeout elapses if the application is still in ACTIVE state.	private, platform
Application-Related (400-599)			
4xx	Application dependent (technical)	These events provide the availability to an application to publish an event to a SM with a bilateral or multilateral agreement on the exact meaning of the event codes.	private

[3] only for output classes

[3] only for output classes

[3] only for output classes

[3] only for output classes

Status Code Description			
Code	**Name**	**Description**	**Event Type**
5xx	Application dependent (security)	These events provide the availability to an application to publish an event to a SM with a bilateral or multilateral agreement on the exact meaning of the event codes.	private
Application State Change Requests (800-899)			
801	CUSS_MANAGER_ REQUEST	Used when CAM sends an event to AL application to change its state upon request from CAM itself.	private, platform
802	SP_SYSTEM_ MANAGER_ REQUEST	Used when CAM sends an event to AL application to change its state upon request from SP System Manager	private, platform
803	AL_SYSTEM_ MANAGER_ REQUEST	Used when CAM sends an event to AL application to change its state upon request from AL System Manager	private, platform
804	CL_APPLICATION_REQU EST	Used when CAM sends an event to AL application to change its state upon request from CLA	private, platform
805	AL_APPLICATION_REQU EST	Used when CAM sends an event about application state changes due to application request.	private, platform
Application-Related (900-999)			
9xx	Application dependent (business, functional)	These events provide the availability to an application to publish an event to a SM with a bilateral or multilateral agreement on the exact meaning of the event codes.	private

Data Status Codes

The Data Status code describes the status of each data record in a MSG data type (for the definition of MSG data stream, refer to Section 3.1.9: Data)

Data Status Code Description		
Code	**Name**	**Description**
0	DS_OK	Data record is OK
1	DS_CORRUPTED	Data record is corrupted (no data included)
2	DS_INCOMPLETE	Data record is incomplete
3	DS_ZEROLENGTH	Data record is of length 0

Please review Chapter 6 for information on how additional DS_TYPES and other data status codes are used for new and extended media types and information.

Please note that if the platform detects and return DS_CORRUPTED as the data status, then it shall not include any data in the data record.

Appx B: Component Mappings

Introduction

It is highly recommended that the number of physical components be minimal. Use a device that can read and write on the same stock rather than using two devices. Use a device that can manage many types of stock, e.g.: magnetic card, ATB2 and chip card, instead of one device per stock type.
The real component represents the real physical peripheral that is installed on a CUSS platform. Many peripherals are mapped into virtual component types to indicate that they have the same general characteristics: media, data type, etc.
A real component may be mapped to one or more set of disjoint virtual components (e.g. A real ATB2 printer could be mapped to a virtual BoardingPass Printer and virtual ReceiptPrinter assuming this ATB2 printer is configured to have both boarding pass and receipt stocks). Refer to Section 2.6.1: Virtual Component Concept for further information.

Application developers should also review Chapter 7, which provides much more extensive information provides much of the same information as here, but organized more practically to assist in writing code to find and use CUSS devices in a kiosk application.

Real Components Mapping

The following table lists typical CUSS real components, their requirement (Mandatory, recommended, optional) and their associated virtual component(s) that each real component consists of.
Depending on the device configuration (device functions and/or media types supported), some real component mapping will need more than one virtual component of the same type but the list includes only one sample of that type. Real components are not listed by company and model number, but by general component, allowing for a reduction in the length of the list and still providing a global view.

Note 1 (from CUSS 1.0 Addendum A.1.28):
This is not a complete list of the types of real devices, of the types of virtual component, or of virtual component linking, which may be found on a kiosk. Specifically, it is not meant as a reference as to how devices can be identified on a kiosk: it does not and cannot reflect every possible CUSS device that might exist on a kiosk.

Most importantly, the “Real Component Name” listed below is not guaranteed to match the realComponentName Characteristic value found for virtual components for that device type in a kiosk. A CUSS application should not rely exclusively on the realComponentName to determine the capabilities and devices that exist on the kiosk. It must analyze the component linking and

characteristics to chose and use the virtual components it needs. Chapter 7 has detailed information on how to do this.

Real Component Name	Requirement	Virtual Component Type /Name
ATB2Device	Recommended	MediaInput
		MediaOutput
		Dispenser
		Feeder
ATB2DeviceWithEscrow	Optional	MediaInput
		MediaOutput
		Dispenser
		Dispenser (for Escrow)
		Capture (for Escrow)
		Feeder
ATB2Printer	Recommended	MediaOutput
		Dispenser
		Feeder
ATB2Reader	Recommended	MediaInput
		Dispenser
		Capture (for Escrow)
Audio	Optional	DataOutput
BagTagPrinter	Recommended	MediaOutput
		Dispenser
		Feeder
BarCodeScanner	Optional	MediaInput
		Capture
BoardingPassCaptureBin	Optional	Capture
BoardingPassDispenserBin	Optional	Dispenser **Or** Feeder
BoardingPassPrinter	Mandatory	MediaOutput
		Dispenser
		Feeder
CardReader (Mag+Chip)	Recommended	MediaInput (for Magnetic cards)
		MediaInput (for Chip cards)
		Dispenser
ChipCardCaptureBin	Optional	Capture
ChipCardDevice	Optional	MediaInput
		MediaOutput
		Dispenser
ChipCardDispenserBin	Optional	Dispenser **Or** Feeder
ChipCardReader	Optional	MediaInput
		Dispenser
ChipCardWriter	Optional	MediaOutput
		Dispenser

Real Component Name	Requirement	Virtual Component Type /Name
		Feeder
Clock	Mandatory	DataInput
		Dispenser
Display	Mandatory	Display
DoorSensor	Recommended	DataInput
Escrow	Optional	Dispenser Capture
FingerprintReader	Optional	UserInput
GPPrinter	Optional	MediaOutput
		Dispenser
		Feeder
HardDisk	Mandatory	Storage
Keypad	Optional	UserInput
LEDIndicator	Optional	UserOutput
MagneticCardDevice	Optional	MediaInput
		MediaOutput
		Dispenser
		Feeder
MagneticCardEncoder	Optional	MediaOutput
		Dispenser
		Feeder
MagneticCardReader	Mandatory	MediaInput
		Dispenser (for motorized readers only)
Monitor	Optional	Display
Network	Mandatory	Network
OCRReader	Optional	MediaInput
PassportReader	Recommended	MediaInput
PinPadEncrypting	Optional	UserInput
		DataOutput
ProximitySensor	Optional	UserInput
RadioRFID	Optional	MediaInput
ReceiptPrinter	Recommended	MediaOutput
		Dispenser
		Feeder
TicketPrinter	Optional	MediaOutput
		Dispenser
		Feeder
		Capture
TouchScreenOverlay	Mandatory	UserInput
UPS	Recommended	DataInput
VideoCamera	Optional	UserInput
VisualCustomerAssistanceLight	Optional	UserOutput
Hardware WatchDog	Recommended	DataInput

Note 2 (taken from CUSS 1.0 Addendum A.1.24):
The list above is not an exhaustive or complete list of all possible virtual device combinations; it is a guide as to the most common types of devices. For example, newer passport readers with features such as physical clamping or full-page scanning may indeed have real Dispenser components.

Application developers must make sure they check for linked components. For example, when reading from a MediaInput device, a linked Dispenser component may exist, in which case the application must call offer() to return documents to the user.

Note 3 (taken from CUSS 1.0 Addendum A.1.17):
If an application needs to play sounds, it shall use native methods and APIs and only play sounds without modifying kiosk behavior such as adjusting the sound volume.

To determine if the kiosk is able to play audio, the application shall look for a virtual Audio component. An application can then play native sounds if and only if a virtual audio component exists. CUSS platforms shall not include this Audio component if the kiosk cannot or should not play audio.

If the audio component exists and an application invokes the send() directive, the platform shall return RC_NOT_SUPPORTED to the application; there is no CUSS-standard way of sending audio to the platform in this fashion.

Appx C: IDL Interface Definition Files

This appendix lists all the CUSS IDL files defining the CUSS CORBA interfaces. This includes:

- **types.idl**: Type definitions used in all CUSS IDL files
- **comps.idl**: Component definition to all CUSS components
- **codes.idl**: Core definitions of all CUSS codes
- **characteristics.idl**: Characteristics definitions of CUSS components
- **CUSS.PAYMENT.XSD**: XML messaging schema for the payment interface
- **CUSS.SBD.XSD**: XML messaging schema for bag tag RFID

Note 1 (from CUSS 1.0 Addendum A.1.33):
Unless specifically addressed within a comment or future Addendum entry, the syntax and type definitions within the IDL override and displace any conflicting passages included elsewhere in this Technical Specification document. For example, if the IDL specifies that a field is an “any” type, but a different section in this document indicates that field is a “name” type (String) then the IDL prevails and the element shall be treated as an “any”.

When and if errors within the IDL itself are found, the conflict will be resolved within the CUSS Technical Group meetings and this technical specification will be updated accordingly.

types.idl (Type definitions for CUSS)

```
//------------------------------------------------------------------------
//
//  File:      types.idl
//
//  Purpose:   Type definitions for CUSS idls
//
//  Date:      17.06.2013
//
//  Version:   1.3
//
//  Author:    IATA Passenger Experience Management Group: CUWG CUSS-TSG
//
//  Copyright(c) 2003,2009,2013 International Air Transport Association, All Rights Reserved
//
//  Note:      Please refer to the CUSS 1.3 Technical Specification for more information
//
//------------------------------------------------------------------------

#ifndef TYPES_IDL
#define TYPES_IDL

#pragma prefix "cuss.iata.org"

/**
 *  Definition of the Data Types
 *
 *  @note  If your version of the IDL compiler treats eventType as a CORBA IDL identifier,
 *         you will need to escape it by prepending an underscore (_) to it, that is
 *         replace all occurrences of eventType with _eventType
 */

module types
{
    typedef string name;                /**< Definition for names */
    typedef sequence<name> namelist;    /**< Definition for name lists */
    typedef sequence<long> indexList;   /**< Definition for a list of indexes */
    typedef string reference;           /**< Used as the application reference (token) */
    typedef string ior;                 /**< CORBA Object reference like IOR:..... */
    typedef sequence<ior> iorlist;      /**< List of IORs */
    typedef sequence<octet> bytestream; /**< Definition for data streams */
    typedef any correlation;            /**< Used as a user defined private identification */

    /** The time out data type.
     * A value > 0 specifies a synchronous call with timeout in MilliSeconds.
     * A value < 0 specifies an asynchronous call with timeout in MilliSeconds
     */
    typedef long timeout;

    const timeout BLOCK_  = 0;       /**< Wait forever on synchronous calls */

    /**
     *  Application and Kiosk Identification
     */
    struct akID
    {
      name companyCode;     /**< eg 3L- or 2L-code for airlines */
      name applicationName; /**< Name of the application */

      name vendorCode;      /**< Vendor specific code (used for SM-Interface) */
      name kioskName;       /**< Name of the kiosk (used for SM-Interface) */
    };

    /**
     * Kiosk location identification
     */
    struct location
    {
      name airportCode;   /**< 3L code for the airport or city or any  location */
      name terminal;      /**< Terminal name, if applicable */
      name area;          /**< Area name, if applicable */
      name address;       /**< Free form address, if applicable */
    };

    /**
```

```
 * Predefinition for GPS
 */
enum orientation
{
    north_,
    south_,
    east_,
    west_,
    undefined_
};

/**
 * Base definition for GPS coordinates
 */
struct coordinate
{
    orientation  direction;   /**< north, south, east, west or undefined */
    long         degrees;     /**< Subdivision in degrees */
    long         minutes;     /**< Subdivision in minutes */
    long         seconds;     /**< Subdivision in seconds */
    long         hundreths;   /**< Subdivision in hundredths of a second */
};

/**
 * CUSS uses GPS coordinates to inform about the exact kiosk location
 */
struct gps
{
    coordinate longitude;  /**< Value for longitude coordinate */
    coordinate latitude;   /**< Value for latitude coordinate */
    long       altitude;   /**< Height in meters from sea level */
};

/**
 * Structure returned with the <i>level-directive</i>
 */
struct EnvironmentLevel
{
    timeout   sessionTimeout;
    /**< session timeout (in milliseconds) for active applications */

    timeout   killTimeout;
    /**< Time (in milliseconds) left before an application is killed */

    akID      kioskID;       /**< Identification of the kiosk */
    location  kioskLocation; /**< Location of the kiosk (text form) */
    gps       gpsLocation;   /**< GPS coordinates of the kiosk */

    name      cussVersion;
    /**< contains a comma-separated string for all CUSS versions supported */

    name      cussInterfaceVersionMin; /**< This field may be left blank */
    name      cussInterfaceVersionMax; /**< This field may be left blank */

    name      jvmName;    /**< Name of the JAVA virtual machine supported */
    name      jvmVersion; /**< Version of the JAVA virtual machine supported */

    name      browserName;    /**< Name of the installed internet browser */
    name      browserVersion; /**< Version of the installed internet browser */

    name      osName;     /**< Name of the installed operating system */
    name      osVersion;  /**< Version of the installed operating system */

    /** Token reference that is passed to applications.
     *  This reference is used as a password for all further directives to the platform */

    reference applicationToken;
};

/**
 * Base environment component definition
 */
struct EnvironmentComponent
{
    name       virtualComponentName;  /**< refer to section 3.2.2  */
    ior        virtualComponentRef;
    /**< CORBA reference to the virtual component (IOR) */

    name       realComponentName;
   /**< This must be unique per peripheral, used for comparison only */
```

```
        indexList  linkedComponents;
        /**< This list of indexes indicates, at which position in the component list
             the linked components can be found (Index counting starts at 0). */

    };

    /**
     * The platform returns a list of all virtual components with this data type
     */
    typedef sequence<EnvironmentComponent> EnvironmentComponents;

    /**
     *  Data-status codes are used to describe the validity of the data records which are
     *  transmitted by an event. The data status codes are defined in file <i>codes.idl</i>
     */
    typedef long dataStatus;

    /**
     * Predefinition used for the CUSS data type definition
     */
    struct dataRecord
    {
        dataStatus  status;  /**< Status of the data in this data record */
        bytestream  message; /**< The data itself */
    };

    /**
     *  The CUSS data type definition.
     *  This data type is used by card readers, passport readers and other devices.
     */
    struct msgDataType
    {
        sequence<dataRecord>  records;   /**< A list of data records */
    };

    /**
     * Type definition for AEA data which is used by ATB2 and BagTag printers
     */
    typedef bytestream aeaDataType;

    /**
     * Type definition for SVG data which is used by a General Purpose Printers (GPP)
     */
    typedef bytestream svgDataType;

    /**
     * Type definition for NULL/NIL data.
     * Indicates that no data is sent by an event
     */
    typedef octet nilDataType;

    /**
     * Definition for the <i>registerEvent</i> directive
     */
    enum action
    {
      subscribe_,   /**< Used to subscribe/register an event */
      discard_      /**< Used to discard/deregister an event */
    };

    /**
     * The category of the event that has been sent
     */
    enum evtCategory
    {
      alarm_,  /**< Manual intervention is required (hard condition) */
      alert_,  /**< Manual intervention is not needed (soft condition)*/
      normal_  /**< Normal event (no error/warning condition) */
    };

    /**
     * The type of the event that has been sent.
  * In CUSS 1.0, if an event is both private and platform, choose platform as the event type
     */
    enum evtType
    {
      invalid_, /**< Invalid event (may be used in the returned event of a directive call */
      private_, /**< Private event (received only by the applicable application) */
      public_,  /**< Public event  (received only by all listening applications)*/
      platform_ /**< Platform event (received only by the applicable application and SP SM) */
    };
```

```
  /**
   * The mode of the event that has been sent
   */
  enum evtMode
  {
    solicited_,   /**< Event is related to a previous directive call */
    unsolicited_  /**< Event is NOT related to any previous directive call */
  };

  /**
   * This definition is used to specify which events should be received by
   * the instance that acquires a component or registers for event(s)
*
   */
  enum evtFilterType
  {
    all_,       /**<  Receive all events */
    any_,       /**<  Receive any event (used for <i>eventWait</i> only) */
    nil_,       /**<  Receive no event */
    code_,      /**<  Receive event related to specific event code(s) */
    type_,      /**<  Receive related to specific event type(s) */
    component_  /**<  Receive related to specific component(s) */
  };

  typedef long evtCode;         /**< Event codes as defined in <i>codes.idl</i> */
  typedef long evtStatusCode;   /**< Status codes as defined in <i>codes.idl</i> */

  /**
   *  This definition is more obvious than just the CORBA::any type.
   *  A datastream may consist of: <i>aeaDataType, svgDataType, nilDataType, msgDataType</i>
   *  (Datastreams must always be complete and NOT segmented).
   *
   *  In CUSS 1.0, datastream may also consist of:
   *  string,        used for clock data type, format is (yyyymmddhhmmss) or
   *  const long,    used for switch data type, value is one the following:
   *  (OFF= 0, ON=1, OPEN=2, CLOSED=3, YES = 4, NO=5, UNKNOWN=6)
   */
  typedef any  datastream;

  /**
   * Predefinition for <i>evtAcquireFilter</i> data type
   */
  union evtCodeFilterUnion switch(evtFilterType)
  {
    case all_       :
    case any_       : any filterALLorANY;
    case component_ : iorlist filterCOMPONENT;
  };

  /**
   * Predefinition for <i>evtAcquireFilter</i> data type
   */
  struct evtCodeFilterElem
  {
    evtCode             eventCode;    /**< The event code for event filtering */
    evtStatusCode       statusCode;   /**< The status code for event filtering */
    evtCodeFilterUnion  eventFilter;  /**< Component filter for the event/status code */
  };

  /**
   * Predefinition for <i>evtAcquireFilter</i> data type
   */
  typedef sequence<evtCodeFilterElem> evtCodeFilter;

  /**
   * Predefinition for <i>evtAcquireFilter</i> data type
   */
  union evtTypeFilterUnion switch(evtFilterType)
  {
    case all_       :
    case any_       : any filterALLorANY;
    case component_ : iorlist filterCOMPONENT;
  };

  /**
   * Predefinition for <i>evtAcquireFilter</i> data type
   */
  struct evtTypeFilterElem
  {
      evtType eventType;              /**< The event type for event filtering */
```

```
      evtTypeFilterUnion eventFilter;  /**< Component filter for the event type */
  };

  /**
   * Predefinition for <i>evtAcquireFilter</i> data type
   */
  typedef sequence<evtTypeFilterElem> evtTypeFilter;

  /**
   * Predefinition for <i>evtAcquireFilter</i> data type
   */
  union evtComponentFilterUnion switch(evtFilterType)
  {
    case all_  :
    case any_  : any filterALLorANY;
    case code_ : sequence<evtCode> filterCODE;
    case type_ : sequence<evtType> filterTYPE;
  };

  /**
   * Predefinition for <i>evtAcquireFilter</i> data type
   */
  struct evtComponentFilterElem
  {
      ior   componentName;                 /**< The component name for event filtering*/
      evtComponentFilterUnion eventFilter;  /**< Event filter for the component name */
  };

  /**
   * Predefinition for <i>evtAcquireFilter</i> data type
   */
  typedef sequence<evtComponentFilterElem> evtComponentFilter;

  /**
   * This data type is passed to the <i>registerEvent</i> directive to specify
   *  which events are received from the platform
*
* @note:   In CUSS 1.0, implementing event filtering is not mandatory.
*
   */
  union evtFilter switch(evtFilterType)
  {
    case all_       :
    case any_       : any filterALLorANY;
    case code_      : evtCodeFilter filterCODE;
    case type_      : evtTypeFilter filterTYPE;
    case component_ : evtComponentFilter filterCOMPONENT;
  };

  /**
   *  This data type is passed to the <i>component acquire</i> directive to specify
   *  which events are received from this virtual component
*
* @note:   In CUSS 1.0, implementing event filtering is not mandatory.
*
   */
  union evtAcquireFilter switch(evtFilterType)
  {
    case all_  :
    case nil_  : any filterALLorNIL;
    case code_ : sequence<evtCode> filterCODE;
    case type_ : sequence<evtType> filterTYPE;
  };

  /**
   * Predefinition for <i>evtDescription</i> data type
   */
  struct evtDescr
  {
      evtCode           eventCode;         /**< Description is related to this event code */
      evtStatusCode     statusCode;        /**< Description is related to this status code */
      sequence<evtType> eventTypes;        /**< Description is related to this event type */
      name              eventDescription;  /**< The textual description of the specified event */
  };

  /**
   * Predefinition for <i>evtDescription</i> data type
   */
  struct evtDescrANY_CODE_TYPE
  {
      evtDescr  eventDescr;     /**< The event description */
```

```
        namelist  componentList;  /**< The event description for these components */
    };

    /**
     * Predefinition for <i>evtDescription</i> data type
     */
    struct evtDescrCOMPONENT
    {
        name               componentName; /**< The event description for this component */
        sequence<evtDescr>  eventDescr;    /**< The event descriptions */
    };

    /**
     * This definition is used to query information about event(s)
     */
    union evtDescription switch(evtFilterType)
    {
      case any_       :
      case code_      :
      case type_      : evtDescrANY_CODE_TYPE eventDescrANY_CODE_TYPE;
      case component_ : evtDescrCOMPONENT eventDescrCOMPONENT;
    };

#ifndef _TIME_BASE_IDL_

    /**
     *  Definition of TimeT borrowed from the CORBA Time Service.
     *  TimeT represents a simple time value, which is 64 bits in size,
     *  and holds the number of 100 nanoseconds that have passed since the base time.
     *  For absolute time calculations, the base is <i>15 October 1582 00:00 </i>.
     *
     *  Note: If your IDL compiler does not yet support the <i>long long</i> data type,
     *        please compile this module with the preprocessor definition <i>NOLONGLONG</i>.
     *
     */

#ifdef NOLONGLONG

    struct ulonglong
    {
        unsigned long low;
        unsigned long high;
    };

    typedef ulonglong TimeT;

#else

    typedef unsigned long long  TimeT;

#endif // NOLONGLONG
#endif // _TIME_BASE_IDL_

    /**
     *  Event definition.
     *  This definition is used for all events and return values that are used within the platform
     */
    struct Event
    {
        TimeT           timeStamp;        /**< Time stamp in UTC format */
        akID            kioskID;          /**< Identification of the kiosk application */
        location        kioskLocation;    /**< Location of the kiosk (text form) */
        gps             gpsLocation;      /**< GPS coordinates of the kiosk */

        name            componentRef;
  /**< reference of the component if it is the event source */

        name            functionName;
        /**< name of the function/directive which has been executed
             (this field will be empty for unsolicited events) */

        evtCode         eventCode;
        /**< Application or component state transition or the current application
             or component state if no transition applies */

        evtMode         eventMode;        /**< solicited, unsolicited */
        evtType         eventType;        /**< invalid, private, public, platform */
        evtCategory     eventCategory;    /**< alarm, alert, normal */
        evtStatusCode   statusCode;       /**< component status or function call status */

        correlation     elud;             /**< user defined private identification */
```

```
        datastream      eventData;        /**< data passed with the event */
    };

    /**
     *  Event listener definition.
     *  This interface is passed on acquiring virtual components or with the
     *  <i>registerEvent</i> directive
     */
    interface evtListener
    {
        /**
         *  This function is called whenever an event is sent to the application.
         *
         *  @param  e    The event that is passed to the application
         */
        void callback(in Event e);
    };
};
#endif // TYPES_IDL
```

comps.idl (Interface to CUSS components)

```
//------------------------------------------------------------------------------
//
//  File:      comps.idl
//
//  Purpose:   Interfaces to CUSS components
//
//  Date:      17.06.2013
//
//  Version:   1.3
//
//  Author:    IATA Passenger Experience Management Group: CUWG CUSS-TSG
//
//  Copyright(c) 2003,2009,2013 International Air Transport Association, All Rights Reserved
//
//  Note:      Please refer to the CUSS 1.3 Technical Specification for more information
//
//------------------------------------------------------------------------------

#ifndef COMPS_IDL
#define COMPS_IDL

#include "codes.idl"
#include "types.idl"
#include "characteristics.idl"

#pragma prefix "cuss.iata.org"

/**
 *  Definition of the interfaces to CUSS Components
 *
 * @note  If your version of the IDL compiler treats Component as a CORBA IDL identifier,
 *        you will need to escape it by prepending an underscore (_) to it , that is
 *        replace all occurrences of Component with _Component
 */

module Components
{
    /**
     *  All components are derived from this interface
     *
     */
    interface Component { };

    /**
     *  All interfaces for peripherals are derived from this interface
     */
    interface CUSSCntl : Component
    {

      /**
       *  Returns the state/status of the virtual component.
       *
       *  @param  to      Timeout value
       *  @param  appRef  A valid application reference (token)
       *  @param  e       Return value
       *
       */
      returncodes::rc query (in  types::timeout to,
                             in  types::reference appRef,
                             out types::Event e);
    };

    /**
     *  Interfaces for virtual components that map to peripheral devices
     */
    interface Peripheral : CUSSCntl
    {

      /**
       *  Make the virtual component available for the application.
       *  The application can subscribe a specific listener associated to the acquired component.
       *
       *  @param  to      Timeout value
```

```
     *  @param  appRef  A valid application reference (token)
     *  @param  ef      Specifies which events to subscribe to (event filter)
     *  @param  el      Specifies the event listener to be set for this component
     *  @param  elud    User data that is submitted with each event sent to the listener
     *  @param  e       Return value
     *
     *  @note   As implementation of event filtering is not required in CUSS 1.0,
     *          event listener passed to acquire will be used as the receiver
     *          for all events related to this component.
     *
     */
    returncodes::rc acquire (in  types::timeout to,
                             in  types::reference appRef,
                             in  types::evtAcquireFilter ef,
                             in  types::evtListener el,
                             in  types::correlation elud,
                             out types::Event e);

    /**
     *  Makes the virtual component unavailable to the application and unsubscribes events
     *  relative to the component.
     *
     *  @param  to      Timeout value
     *  @param  appRef  A valid application reference (token)
     *  @param  e       Return value
     *
     */
    returncodes::rc release (in  types::timeout to,
                             in  types::reference appRef,
                             out types::Event e);

    /**
     *  Set up the virtual component and its profile for the application.
     *
     *  @param  to      Timeout value
     *  @param  appRef  A valid application reference (token)
     *  @param  ds      Datastream for setting the component (e.g. PECTABS)
     *  @param  e       Return value
     *
     */
    returncodes::rc setup (in  types::timeout to,
                           in  types::reference appRef,
                           in  types::datastream ds,
                           out types::Event e);

    /**
     *  Allows to cancel all pending (previously called in asynchronous mode)
     *  directives on this specific component.
     *
     *  @param  appRef  A valid application reference (token)
     *  @param  e       Return value
     *
     */
    returncodes::rc cancel (in  types::reference appRef,
                            out types::Event e);

    /**
     *  Test the virtual and real component as exhaustive as possible.
     *  If the component is a physical device the device driver should be accessed but
     *  the physical device should not be exercised.
     *
     *  @param  appRef            A valid application reference (token)
     *  @param  e                 Return value
     *
     */
    returncodes::rc test (in  types::timeout to,
                          in  types::reference appRef,
                          out types::Event e);
};

/**
 *  Interface for virtual components that provide data to applications
 */
interface Input
{

    /**
     *  Make the data from the virtual component available to the application.
     *
     *  @param  to      Timeout value
     *  @param  appRef  A valid application reference (token)
```

```
     *  @param  e       Return value
     *
     */
    returncodes::rc receive (in  types::timeout to,
                             in  types::reference appRef,
                             out types::Event e);
};

/**
 *  Interface for virtual components that are able to receive data from applications
 */
interface Output
{

    /**
     *  Send data from the application to the virtual component.
     *
     *  @param  to      Timeout value
     *  @param  appRef  A valid application reference (token)
     *  @param  e       Return value
     *
     */
    returncodes::rc send (in  types::timeout to,
                          in  types::reference appRef,
                          in  types::datastream ds,
                          out types::Event e);
};

/**
 *  Interface for virtual components that interact with customers/users
 */
interface User
{

    /**
     *  Make the virtual component available for the user.
     *  (e.g. enables a CardReader device for card insertion)
     *
     *  @param  to      Timeout value
     *  @param  appRef  A valid application reference (token)
     *  @param  e       Return value
     *
     */
    returncodes::rc enable (in  types::timeout to,
                            in  types::reference appRef,
                            out types::Event e);

    /**
     *  Makes the virtual component unavailable for the user.
     *  (e.g. disables a CardReader device from card insertion)
     *
     *  @param  to      Timeout value
     *  @param  appRef  A valid application reference (token)
     *  @param  e       Return value
     *
     */
    returncodes::rc disable (in  types::timeout to,
                             in  types::reference appRef,
                             out types::Event e);
};

/**
 *  Interface for peripherals that don't interact with users/customers
 */
interface Userless { };

/**
 *  Interface for virtual components that use a physical media
 *  (e.g. card, coupon, or a paper document)
 */
interface Media { };

/**
 *  Interface for virtual components that don't use a physical media
 *  (e.g. card, coupon, or a paper document)
 */
interface Medialess { };

/**
 *  Interface for virtual components that transfer data
 */
```

```
interface Data { };

/**
 *  Interface for virtual components that don't transfer data
 */
interface Dataless { };

/**
 *  Interface for virtual components that are able to retain media
 */
interface Capture : Peripheral, Userless, Media, Dataless, Characteristics::Capture
{

  /**
   *  Captures the document in the virtual component that is associated to a secured bin.
   *
   *  @param  to      Timeout value
   *  @param  appRef  A valid application reference (token)
   *  @param  e       Return value
   *
   */
  returncodes::rc retain (in  types::timeout to,
                          in  types::reference appRef,
                          out types::Event e);
};

/**
 *  Interface for virtual components that receive media from a Peripheral component
 *  and offer it to the user or to another Peripheral component
 *  (e.g. ejecting an ATB coupon from the printer to the ESCROW)
 */
interface Dispenser : Peripheral, User, Media, Dataless, Characteristics::Dispenser
{

  /**
   *  Offer the document from the virtual component to the user or to another component.
   *
   *  @param  to      Timeout value
   *  @param  appRef  A valid application reference (token)
   *  @param  e       Return value
   *
   */
  returncodes::rc offer (in  types::timeout to,
                         in  types::reference appRef,
                         out types::Event e);
};

/**
 *  Interface for virtual components that are holding media (e.g. ATB stocks)
*  and supply it to another Peripheral component
 */
interface Feeder : Peripheral, Userless, Media, Dataless, Characteristics::Feeder
{

  /**
   *  Offer the document from a feeder to another virtual component.
   *
   *  @param  to      Timeout value
   *  @param  appRef  A valid application reference (token)
   *  @param  e       Return value
   *
   */
  returncodes::rc offer (in  types::timeout to,
                         in  types::reference appRef,
                         out types::Event e);
};

/**
 *  Interface for virtual components used for inbound data transfer (e.g. digital input)
 */
interface DataInput : Peripheral, Userless, Medialess, Data, Input, Characteristics::DataInput { };

/**
 *  Interface for virtual components used for outbound data transfer (e.g. network output)
 */
interface DataOutput : Peripheral, Userless, Medialess, Data, Output, Characteristics::DataOutput { };

/**
 *  Interface for virtual components used for inbound user data transfer (e.g. sound device)
 *
 *  @note    For the touch screen overlay, which is a native device implemented as UserInput,
```

```
 *           Only acquire, release and query directives should be implemented.
 *           All other inherited methods are not applicable and should return RC_NOT_SUPPORTED
 *
 */
interface UserInput : Peripheral, User, Medialess, Data, Input, Characteristics::UserInput { };

/**
 *  Interface for virtual components used for outbound user data transfer (e.g. screen)
 */
interface UserOutput : Peripheral, User, Medialess, Data, Output, Characteristics::UserOutput { };

/**
 *  Interface for virtual components used for reading from media (e.g. mag card reader)
 */
interface MediaInput : Peripheral, User, Media, Data, Input, Characteristics::MediaInput { };

/**
 *  Interface for virtual components used for writing to media (e.g. receipt printer)
 */
interface MediaOutput : Peripheral, User, Media, Data, Output, Characteristics::MediaOutput { };

/**
 *  Interface for virtual components used for reading/writing from/to storage (e.g. harddisk)
 *
 *  @note    As Storage is a native device,
 *           only acquire, release and query directives should be implemented.
 *           All other inherited methods are not applicable and should return RC_NOT_SUPPORTED
 *
 */
interface Storage : Peripheral, Userless, Media, Characteristics::Storage { };

/**
 *  Interface for virtual components handling a display (eg. kiosk computer screen)
 *
 *  @note    As Display is a native device,
 *           only acquire, release and query directives should be implemented.
 *           All other inherited methods are not applicable and should return RC_NOT_SUPPORTED
 *
 */
interface Display : Peripheral, User, Medialess, Characteristics::Display { };

/**
 *  Interface for virtual components handling network access
 *
 *  @note    As Network is a native device,
 *           only acquire, release and query directives should be implemented.
 *           All other inherited methods are not applicable and should return RC_NOT_SUPPORTED
 *
 */
interface Network : Peripheral, Userless, Medialess, Characteristics::Network { };

/**
 *  Interface for virtual components representing a stand-alone baggage scale
 */

interface BaggageScale : UserInput, Characteristics::BaggageScale { };

/**
 *  Base definition for virtual components allowing self-service baggage check-in
*  @note These are the new definitions for CUSS 1.3. Older definitions are deprecated
*  from CUSS version 1.3 and will be completely removed in CUSS 1.5 latest.
 */

interface ConveyorSBD : Peripheral, Medialess, Output, Input, Characteristics::ConveyorSBD
{
    /**
     *  Moves current piece of baggage to next position. The next position can
     *  be the airports take-away belt (dispatching baggage).
     *
     *  @param  to      Timeout value
     *  @param  appRef  A valid active application reference
     *  @param  e       Return value
     */

    returncodes::rc forward (in  types::timeout to,
                             in  types::reference appRef,
                             out types::Event e);

    /**
     *  Moves current piece of baggage back to the previous postion/user.
     *
```

```
         *  @param  to      Timeout value
         *  @param  appRef  A valid active application reference
         *  @param  e       Return value
         */

        returncodes::rc backward (in  types::timeout to,
                                  in  types::reference appRef,
                                  out types::Event e);

        /**
         *  Processes current piece of baggage on the conveyor.
         *  Allows applications to execute a verification without
         *  physically moving a bag back and/or forth.
         *  May return RC_NOT_SUPPORTED, if a component does not support
         *  or implement that function
         *
         *  @param  to      Timeout value
         *  @param  appRef  A valid active application reference
         *  @param  e       Return value
         */

        returncodes::rc process  (in  types::timeout to,
                                  in  types::reference appRef,
                                  out types::Event e);
    };

    /**
     *  Interface for virtual components representing the baggage insertion position of a conveyor system
    *
    *  @note To better reflect the process of baggage check-in,
    *  comprising of insertion and weighing, verification and waiting for a free slot on the carry-off belt
    *  the definition of the Integrated Baggage System always has three conveyor segments
    *  InsertionBelt, VerificationBelt and ParkingBelt (...even when there is no physical representation of
e.g. a verification belt)
     */

    interface InsertionBelt : ConveyorSBD, User
    {
      /**
        *  Offer the bag from the virtual component (back) to the user
        *  (Define own offer() to avoid multiple inheritance)
        *
          */

        returncodes::rc offer (in  types::timeout to,
                        in  types::reference appRef,
                        out types::Event e);
   };

    /**
     *  Interface for virtual components representing the baggage verification position of a conveyor system
    *
    *  @note To better reflect the process of baggage check-in,
    *  comprising of insertion and weighing, verification and waiting for a free slot on the carry-off belt
    *  the definition of the Integrated Baggage System always has three conveyor segments
    *  InsertionBelt, VerificationBelt and ParkingBelt (...even when there is no physical representation of
e.g. a verification belt)
    *
     */

    interface VerificationBelt : ConveyorSBD, Userless
   {
   };

    /**
     *  Interface for virtual components representing the baggage parking position of a conveyor system
    *
    *  @note To better reflect the process of baggage check-in,
    *  comprising of insertion and weighing, verification and waiting for a free slot on the carry-off belt
    *  the definition of the Integrated Baggage System always has three conveyor segments
    *  InsertionBelt, VerificationBelt and ParkingBelt (...even when there is no physical representation of
e.g. a verification belt)
    *
     */

    interface ParkingBelt : ConveyorSBD, Userless
   {
   };

    /**
     *  Interface for virtual components which are able to transport baggage
```

```
*  @note This interface definition is deprecated from CUSS 1.3. Application-suppliers
* are encouraged to implement the interfaces InsertionBelt, VerificationBelt and ParkingBelt
* for self-service baggage check-in support.
 */

interface Conveyor : Peripheral, User, Medialess, Data, Input, Characteristics::Conveyor
{
    /**
     *  Moves current piece of baggage to the parking position.
     *
     *  @param  to      Timeout value
     *  @param  appRef  A valid active application reference
     *  @param  e       Return value
     */

    returncodes::rc accept (in  types::timeout to,
                            in  types::reference appRef,
                            out types::Event e);

    /**
     *  Moves current piece of baggage back to the user.
     *
     *  @param  to      Timeout value
     *  @param  appRef  A valid active application reference
     *  @param  e       Return value
     */

    returncodes::rc reject (in  types::timeout to,
                            in  types::reference appRef,
                            out types::Event e);


    /**
     *  Moves baggage from the parking position to the airport's baggage system.
     *
     *  @param  to      Timeout value
     *  @param  appRef  A valid active application reference
     *  @param  e       Return value
     */

    returncodes::rc forwardParked (in  types::timeout to,
                                   in  types::reference appRef,
                                   out types::Event e);


    /**
     *  Moves all baggage from the parking position back to the user.
     *
     *  @param  to      Timeout value
     *  @param  appRef  A valid active application reference
     *  @param  e       Return value
     */

    returncodes::rc returnParked (in  types::timeout to,
                                  in  types::reference appRef,
                                  out types::Event e);

    /**
     *  Allows the user to take back his baggage.
     *
     *  @param  to      Timeout value
     *  @param  appRef  A valid active application reference
     *  @param  e       Return value
     */

    returncodes::rc waitForRemovedBaggage (in  types::timeout to,
                                           in  types::reference appRef,
                                           out types::Event e);

};

/**
  * This interface is used to query the state and/or characteristics
  * of a kiosk application that is configured on the platform.
  *
  */
interface Application : CUSSCntl, Characteristics::Application { };

/**
 *  This interface is inherited by the ApplicationManager and the SystemProviderInterface
 */
```

```
interface ManagementInterface : Component
{

  /**
   *  This is the first directive to be issued by the application
   *  to get basic information on the specific CUSS Platform implementation
   *  If the application is known by the platform (via configuration),
   *  the application reference (token) is returned with this call.
   *
   *  @param  appid     Application identifier which must be configured within the platform
   *                    (for minimal security)
   *  @param  el        Return values
   *
   */
  returncodes::rc level (in  types::akID appid,
                         out types::EnvironmentLevel el);

  /**
   *  This is the second directive to be issued by an application
   *  to get the list of all implemented CUSS components.
   *
   *  @param  appRef  A valid application reference (token)
   *  @param  ec      List of virtual components which contains the CORBA references (IORs)
   *
   */
  returncodes::rc components (in  types::reference appRef,
                              out types::EnvironmentComponents ec);

  /**
   *  Allows applications to wait for an event to occur.
   *  To wait for an event, the application must have subscribed to it via the acquire or
   *  registerEvent directives. The waitEvent-directive will be completed at event occurrence
   *  (any or all in the list) or when the timeout expires.
   *
   *  @param  to      Timeout value (positive and negative values have the same effect for this directive)
   *
   *  @param  appRef  A valid application reference (token)
   *  @param  ef      Specifies the event(s) to wait for
   *  @param  e       Return value
   *
   *  @note   In CUSS 1.0, implementing event filtering is not mandatory.
   */
  returncodes::rc waitEvent (in  types::timeout to,
                             in  types::reference appRef,
                             in  types::evtFilter ef,
                             out types::Event e);

  /**
   *  Generate an event to a system manager.
   *
   *  @param  appRef  A valid application reference (token)
   *  @param  ie      Event to be generated
   *  @param  oe      Return value
   *
   */
  returncodes::rc generateEvent(in  types::reference appRef,
                                in  types::Event ie,
                                out types::Event oe);

  /**
   *  Returns a description of an event.
   *
   *  @param  appRef  A valid application reference (token)
   *  @param  ef      Specifies the event(s) to query
   *  @param  ed      Returned event description(s)
   *
   *  @note   In CUSS 1.0, the implementation of queryEvent is not mandatory.
   *          In this case, this function should return RC_NOT_SUPPORTED.
   */
  returncodes::rc queryEvent (in  types::reference appRef,
                              in  types::evtFilter ef,
                              out types::evtDescription ed);

  /**
   *  Subscribe to or discards from receiving any related event notification. The use
   *  of this directive has an additive effect, which means that a call will not supersede
   *  a previous call but, instead, subscribe for previous event(s) plus the one(s) in the
   *  current call. All subscriptions done with this directive will be received, within the
   *  application, via a single listener.
   *
   *  @param  appRef  A valid application reference (token)
```

```
     *  @param  act     Either subscribe or discard event(s)
     *  @param  ef      Specifies the event(s) to register i.e. event filter
     *  @param  el      Specifies the event listener to be notified
     *  @param  elud    User data that is submitted to the listener on each invocation
     *  @param  ed      Return value
     *
     *  @note   As implementation of event filtering is not required in CUSS 1.0,
     *          event listener passed to registerEvent directive will be used as the
     *          receiver for all application manager events. To receive component events,
     *          application must register their listener(s) via the acquire directive.
     */
    returncodes::rc registerEvent(in  types::reference appRef,
                                  in  types::action act,
                                  in  types::evtFilter ef,
                                  in  types::evtListener el,
                                  in  types::correlation elud,
                                  out types::Event e);

};

/**
 *  The definition for the Application Manager Interface.
 *  An application uses this interface for interaction with the platform.
 *  To access the platform use: <i>corbaloc:<kiosk-IP address>:20000/ApplicationManager</i>
 */
interface ApplicationManager : ManagementInterface
{
    /**
     *  The application now wants to (re-)initialize. This is a blocking call.
     *  After this directive returns the application is allowed to initialize.
     *  This handling ensures that initialization is serialized for all applications.
     */
    returncodes::rc initrequest (in  types::reference appRef,
                                 out types::Event e);

    /**
     *  This directive is used by the application to request a state change from
     *  CUSS Application Manager, which will change the application state if request is approved.
     *
     *  @param  appRef  A valid application reference (token)
     *  @param  ie      Input of application state transition code
     *  @param  oe      Return value
     *
     */
    returncodes::rc notify (in  types::reference appRef,
                            in  types::akID   id,
                            in  types::evtCode ec,
                            out types::Event   e);
};

/**
 *  The definition for the System Manager Interface.
 *  A System Manager Application uses this interface for interaction with the platform
 *  To access the platform use: <i>corbaloc:<kiosk-IP address>:20001/ServiceProviderInterface</i>
 *
 *  @note   In CUSS 1.0, the ServiceProviderInterface is available for both
 *          Service Provider System Manager and Application Provider System Manager
 */
interface ServiceProviderInterface : ManagementInterface
{

    /**
     *  Ask CUSS application manager to load an application
     *  (realize Load state transition in application state diagram).
     *
     *  @param  to       Timeout value
     *  @param  appRef   A valid application reference (token)
     *  @param  whichApp The identifier of the application to be loaded
     *  @param  e        Return value
     *
     */
    returncodes::rc load (in  types::timeout to,
                          in  types::reference appRef,
                          in  types::akID whichApp,
                          out types::Event e);

    /**
     *  Suspend an application.
     *
     *  @param  appRef   A valid application reference (token)
     *  @param  whichApp The identification of the application to be suspended
```

```
       *  @param  e        Return value
       *
       */
      returncodes::rc suspend (in  types::reference appRef,
                               in  types::akID whichApp,
                               out types::Event e);

      /**
       *  Suspend all applications.
       *
       *  @param  appRef   A valid application reference (token)
       *  @param  e        Return value
       *
       */
      returncodes::rc suspendAll (in types::reference appRef,
                                  out types::Event e);

      /**
       *  Resume a suspended application to its previous state.
       *
       *  @param  appRef   A valid application reference (token)
       *  @param  whichApp The identifier of the application to be resumed
       *  @param  e        Return value
       *
       */
      returncodes::rc resume (in  types::reference appRef,
                              in  types::akID whichApp,
                              out types::Event e);

      /**
       *  Resume all suspended applications to their previous state.
       *
       *  @param  appRef   A valid application reference (token)
       *  @param  e        Return value
       *
       */
      returncodes::rc resumeAll (in  types::reference appRef,
                                 out types::Event e);

      /**
       *  Stops (unloads) an application.
       *
       *  @param  appRef   A valid application reference (token)
       *  @param  whichApp The identifier of the application to be stopped
       *  @param  e        Return value
       *
       */
      returncodes::rc stop (in  types::reference appRef,
                            in  types::akID whichApp,
                            out types::Event e);

      /**
       *  Stops (unloads) all applications.
       *
       *  @param  appRef   A valid application reference (token)
       *  @param  e        Return value
       *
       */
      returncodes::rc stopAll (in  types::reference appRef,
                               out types::Event e);
    };
};
#endif  // COMPS_IDL
```

codes.idl (Definitions of CUSS codes)

```
//------------------------------------------------------------------------
//
//  File:      codes.idl
//
//  Purpose:   Definition of CUSS codes
//
//  Date:      17.06.2013
//
//  Version:   1.3
//
//  Author:    IATA Passenger Experience Management Group: CUWG CUSS-TSG
//
//  Copyright(c) 2003,2009,2013 International Air Transport Association, All Rights Reserved
//
//  Note:      Please refer to the CUSS 1.3 Technical Specification for more information
//
//------------------------------------------------------------------------

#ifndef CODES_IDL
#define CODES_IDL

#pragma prefix "cuss.iata.org"

/** Directive related return codes */

module returncodes
{
    typedef long rc; /**< Type definition for interface return codes */

    const long RC_OK            =  0;  /**< Directive accepted */
    const long RC_REFERENCE     = -1;  /**< Invalid application reference */
    const long RC_STATE         = -2;  /**< Application is not in the correct state to invoke this directive */
    const long RC_DENIED        = -3;  /**< Access denied (application is not allowed to use that component) */
    const long RC_PARAMETER     = -4;  /**< Error in parameters (e.g. wrong event passed) */
    const long RC_ANY_PARAMETER = -5;  /**< Error in CORBA::any type */
    const long RC_LISTENER      = -6;  /**< No listener set */
    const long RC_SHARE         = -7;  /**< Request in wrong share mode (component may be blocked by any
application) */
    const long RC_UNAUTHORIZED  = -8;  /**< Unauthorized command within data stream (SVG or AEA) */
    const long RC_ERROR         = -9;  /**< Any error that is not covered by errors defined above */
    const long RC_NOT_SUPPORTED = -10; /**< Directive is not supported (i.e. not implemented) */
};

/** Data related states */

module datastatus
{
    const long DS_OK                            = 0;     /**< Data is OK */
    const long DS_CORRUPTED                     = 1;     /**< Data is corrupted */
    const long DS_INCOMPLETE                    = 2;     /**< Data is incomplete */
    const long DS_ZEROLENGTH                    = 3;     /**< Data is of length zero */

    const long DS_DOCUMENT_AUTHENTICATION_FAILED = 4;    /**< Authentication of document data failed */
    const long DS_INVALID                       = 5;     /**< Document read but a security feature is missing
*/
    const long DS_MISMATCH                      = 6;     /**< Document read but data inconsistent with
security feature */

    const long DS_TYPES_FOID_ISO                = 100;   /**< ISO track data with FOID Data truncation */
    const long DS_TYPES_PAYMENT_ISO             = 200;   /**< ISO track data without truncation */
    const long DS_TYPES_DISCRETIONARY_ISO       = 300;   /**< ISO track data with DISCRETIONARY Data
truncation */

    const long DS_TYPES_FOID_JIS2               = 14100; /**< JIS-2 track data with FOID Data truncation */
    const long DS_TYPES_PAYMENT_JIS2            = 14200; /**< JIS-2 track data without truncation */
    const long DS_TYPES_DISCRETIONARY_JIS2      = 14300; /**< JIS-2 track data with DISCRETIONARY Data
truncation */

    const long DS_TYPES_ISO                     = 0;     /**< ISO encoded data */
    const long DS_TYPES_VING                    = 1000;  /**< VING encoded data */
    const long DS_TYPES_TESSA                   = 2000;  /**< TESSA encoded data */
    const long DS_TYPES_SAFLOK                  = 3000;  /**< SAFLOK encoded data */
    const long DS_TYPES_TIMELOX                 = 4000;  /**< TIMELOC encoded data */
    const long DS_TYPES_KABA_ILCO               = 5000;  /**< KABA iLco encoded data */
```

```
    const long DS_TYPES_KABA_ILCO_FOLIO          = 6000;  /**< KABA iLco (folio) encoded data */

    const long DS_TYPES_IMAGE_IR                 = 7000;  /**< Infrared image */
    const long DS_TYPES_IMAGE_VIS                = 8000;  /**< Visible image */
    const long DS_TYPES_IMAGE_UV                 = 9000;  /**< Ultraviolet image */
    const long DS_TYPES_IMAGE_PHOTO              = 10000; /**< Photo image */
    const long DS_TYPES_IMAGE_COAX               = 11000; /**< Coaxial image */
    const long DS_TYPES_CODELINE                 = 12000; /**< Codeline data */
    const long DS_TYPES_BARCODE                  = 13000; /**< Barcode data */
    const long DS_TYPES_MIWA                     = 14000; /**< Miwa data */
    const long DS_TYPES_JIS2                     = 14000; /**< JIS2 data */

    const long DS_TYPES_SCAN_PDF417              = 15000; /**< PDF417     2D barcode */
    const long DS_TYPES_SCAN_AZTEC               = 15100; /**< Aztec      2D barcode */
    const long DS_TYPES_SCAN_DMATRIX             = 15200; /**< Datamatrix 2D barcode */
    const long DS_TYPES_SCAN_QR                  = 15300; /**< QR Code    2D barcode */
    const long DS_TYPES_SCAN_CODE39              = 15400; /**< Code39     1D barcode */
    const long DS_TYPES_SCAN_CODE128             = 15500; /**< Code128    1D barcode */
    const long DS_TYPES_SCAN_CODE2OF5            = 15600; /**< Code2of5   1D barcode */

    const long DS_TYPES_ISO7816                  = 16000; /**< Communication protocols for PICC/RFID/NFC
devices */

    const long DS_TYPES_PRINT_2S_PAGE            = 16100; /**< 2-Sided Single-page printing */
    const long DS_TYPES_PRINT_2S_MULTI           = 16200; /**< 2-Sided Multi-page printing */
    const long DS_TYPES_PRINT_PDF                = 16300; /**< Adobe PDF print format */

    const long DS_TYPES_MIFARE                   = 17000; /**< Communication protocols for PICC/RFID/NFC
devices */
    const long DS_TYPES_SUICA                    = 17010; /**< Communication protocols for PICC/RFID/NFC
devices */

    const long DS_TYPES_ISO15961                 = 18000; /**< IATA RFID baggage tag devices */
    const long DS_TYPES_RP1745                   = 18010; /**< IATA Baggage Service Messages Format */
    const long DS_TYPES_WEIGHT                   = 18020; /**< Baggage Weight from Scale or Conveyor */
    const long DS_TYPES_HEAVYTAG                 = 18030; /**< Special Heavy Tag for baggage */
    const long DS_TYPES_SBDAEA                   = 18040; /**< AEA-SBD control language */
    const long DS_TYPES_SBDCUSS                  = 18050; /**< CUSS-SBD control language */

    const long DS_TYPES_EPASSPORT_DG1            = 20100; /**< e-Passport format */
    const long DS_TYPES_EPASSPORT_DG2            = 20200; /**< e-Passport format */
    const long DS_TYPES_EPASSPORT_DG3            = 20300; /**< e-Passport format */
    const long DS_TYPES_EPASSPORT_DG4            = 20400; /**< e-Passport format */
    const long DS_TYPES_EPASSPORT_DG5            = 20500; /**< e-Passport format */
    const long DS_TYPES_EPASSPORT_DG6            = 20600; /**< e-Passport format */
    const long DS_TYPES_EPASSPORT_DG7            = 20700; /**< e-Passport format */
    const long DS_TYPES_EPASSPORT_DG8            = 20800; /**< e-Passport format */
    const long DS_TYPES_EPASSPORT_DG9            = 20900; /**< e-Passport format */
    const long DS_TYPES_EPASSPORT_DG10           = 21000; /**< e-Passport format */
    const long DS_TYPES_EPASSPORT_DG11           = 21100; /**< e-Passport format */
    const long DS_TYPES_EPASSPORT_DG12           = 21200; /**< e-Passport format */
    const long DS_TYPES_EPASSPORT_DG13           = 21300; /**< e-Passport format */
    const long DS_TYPES_EPASSPORT_DG14           = 21400; /**< e-Passport format */
    const long DS_TYPES_EPASSPORT_DG15           = 21500; /**< e-Passport format */
    const long DS_TYPES_EPASSPORT_DG16           = 21600; /**< e-Passport format */
    const long DS_TYPES_EPASSPORT_DG17           = 21700; /**< e-Passport format */
    const long DS_TYPES_EPASSPORT_DG18           = 21800; /**< e-Passport format */
    const long DS_TYPES_EPASSPORT_DG19           = 21900; /**< e-Passport format */
    const long DS_TYPES_EPASSPORT_DG20           = 22000; /**< e-Passport format */

    const long DS_TYPES_EPAYMENT                 = 23000; /**< E-Payment: CuPaySchema.xsd */
};

/**
 *  Event codes for all components and applications.
 *  These codes are used to indicate the states or state changes for all components
 */

module eventcodes
{
    // virtual component state transition codes

    const long EC_OK                           = 0; /**< Used in the returned event for calls to suspendAll,
resumeAll or stopAll directives */
    const long EVENTHANDLING_READY             = 1; /**< Used for soft conditions and Ok only */
    const long UNAVAILABLE_RELEASED_PLATFORM   = 2; /**< Released by any authorized platform component */
    const long EVENTHANDLING_UNAVAILABLE       = 3; /**< Caused by a hard condition */
    const long UNAVAILABLE_RELEASED_APPLICATION = 4; /**< Component released by the application */
    const long READY_RELEASED_APPLICATION      = 5; /**< Component released by the application */
    const long READY_RELEASED_PLATFORM         = 6; /**< Released by any authorized platform component */
    const long RELEASED_READY                  = 7; /**< State change caused by a call to <i>acquire</i> */
```

```
    const long RELEASED_UNAVAILABLE             = 8; /**< State change caused by a call to <i>acquire</i> */

    // application state transition codes

    const long INITIALIZE_DISABLED              = 101; /**< State transition DISABLE */
    const long AVAILABLE_DISABLED               = 102; /**< State transition DISABLE */
    const long ACTIVE_DISABLED                  = 103; /**< State transition DISABLE */
    const long UNAVAILABLE_AVAILABLE            = 104; /**< State transition WAIT */
    const long AVAILABLE_ACTIVE                 = 105; /**< State transition ACTIVATE*/
    const long ACTIVE_AVAILABLE                 = 106; /**< State transition WAIT */
    const long INITIALIZE_STOPPED_STOP          = 107; /**< State transition STOP */
    const long AVAILABLE_STOPPED_STOP           = 108; /**< State transition STOP */
    const long ACTIVE_STOPPED_STOP              = 109; /**< State transition STOP */
    const long SUSPENDED_STOPPED_STOP           = 110; /**< State transition STOP */
    const long DISABLED_STOPPED_STOP            = 111; /**< State transition STOP */
    const long SUSPENDED_AVAILABLE              = 112; /**< State transition RESUME*/
    const long AVAILABLE_SUSPENDED              = 113; /**< State transition SUSPEND */
    const long INITIALIZE_STOPPED_RESTART       = 114; /**< State transition RESTART */
    const long AVAILABLE_STOPPED_RESTART        = 115; /**< State transition RESTART */
    const long ACTIVE_STOPPED_RESTART           = 116; /**< State transition RESTART */
    const long DISABLED_STOPPED_RESTART         = 117; /**< Not used in CUSS 1.0. */
    const long SUSPENDED_STOPPED_RESTART        = 118; /**< State transition RESTART */
    const long STOPPED_INITIALIZE               = 119; /**< State transition LOAD */
    const long DISABLED_INITIALIZE              = 120; /**< State transition LOAD */
    const long UNAVAILABLE_STOPPED_RESTART      = 121; /**< State transition RESTART */
    const long UNAVAILABLE_DISABLED             = 122; /**< State transition DISABLE */
    const long UNAVAILABLE_SUSPENDED            = 123; /**< State transition SUSPEND */
    const long INITIALIZE_SUSPENDED             = 124; /**< Not used in CUSS 1.0 */
    const long SUSPENDED_DISABLED               = 125; /**< Not used in CUSS 1.0 */
    const long SUSPENDED_INITIALIZE             = 126; /**< Not used in CUSS 1.0 */
    const long SUSPENDED_UNAVAILABLE            = 127; /**< State transition RESUME */
    const long UNAVAILABLE_STOPPED_STOP         = 128; /**< State transition STOP */
    const long INITIALIZE_UNAVAILABLE           = 129; /**< State transition CHECK */
    const long AVAILABLE_UNAVAILABLE            = 130; /**< State transition CHECK */
    const long DISABLED_SUSPENDED               = 131; /**< Not used in CUSS 1.0 */

    // Application/component state codes

    const long RELEASED                         = 201; /**< State RELEASED (peripheral) */
    const long UNAVAILABLE                      = 202; /**< State UNAVAILABLE (peripheral & application) */
    const long READY                            = 203; /**< State READY (peripheral) */
    const long STOPPED                          = 204; /**< State STOPPED (application) */
    const long SUSPENDED                        = 205; /**< State SUSPENDED (application) */
    const long DISABLED                         = 206; /**< State DISABLED (application) */
    const long INITIALIZE                       = 207; /**< State INITIALIZE (application) */
    const long AVAILABLE                        = 208; /**< State AVAILABLE (application) */
    const long ACTIVE                           = 209; /**< State ACTIVE (application) */
    const long BUSY                             = 210; /**< Transient state BUSY (peripheral) */

    // Additional codes for CUSS 1.2

    const long ACTIVE_ACTIVE                    = 132;  /**< State transition ACTIVE (persistent) */
    const long STATE_EXPLANATION                = 1000; /**< Tell CLA why application is in current state */
    const long ACTIVE_TRANSFER                  = 1001; /**< Request CLA transfer ACTIVE to new application */
    const long TRANSACTION_EXPLANATION          = 1002; /**< Tell CLA what happened in most recent transaction
*/
    const long VERSION_EXPLANATION              = 1003; /**< Tell CLA the version string for the application */

    // Additional codes for CUSS 1.3

    const long ACTIVE_UNAVAILABLE               = 133;  /**< Go unavailable while active in a transaction */
    const long UPDATE_REQUEST                   = 1004; /**< Tell CLA the application wishes to update */

};

/**
 *  Status codes for all virtual device components.
 *  These codes are used to describe the states for all virtual components
 */

module statuscodes
{
    const long OK                       =   0; /**< <b>Scope:</b> public (private for solicited events) */
    const long TIMEOUT                  =   1; /**< <b>Scope:</b> private (directive related) */
    const long WRONG_STATE              =   2; /**< <b>Scope:</b> private + platform */
    const long CANCELLED                =   3; /**< <b>Scope:</b> private */
    const long SOFTWARE_ERROR           =   4; /**< <b>Scope:</b> private + platform */
    const long ALMOST_OUT_OF_TIME       =   5; /**< @note NOT used in CUSS 1.0 */
    const long OUT_OF_SEQUENCE          =   6; /**< <b>Scope:</b> private */

    const long MEDIA_JAMMED             = 101; /**< <b>Scope:</b> public */
```

```
    const long MEDIA_MISPLACED            = 102; /**< <b>Scope:</b> private + platform (platform only for
userless classes) */
    const long MEDIA_PRESENT              = 103; /**< <b>Scope:</b> private */
    const long MEDIA_ABSENT               = 104; /**< <b>Scope:</b> private */
    const long MEDIA_HIGH                 = 105; /**< <b>Scope:</b> public */
    const long MEDIA_FULL                 = 106; /**< <b>Scope:</b> public */
    const long MEDIA_LOW                  = 107; /**< <b>Scope:</b> public */
    const long MEDIA_EMPTY                = 108; /**< <b>Scope:</b> public */
    const long MEDIA_DAMAGED              = 109; /**< <b>Scope:</b> public */
    const long MEDIA_INCOMPLETELY_INSERTED = 110; /**< <b>Scope:</b> private */

    // definitions for baggage belt

    const long BAGGAGE_FULL                 = 120; /**< <b>Scope:</b> public */
    const long BAGGAGE_UNDETECTED           = 121; /**< <b>Scope:</b> private + platform */
    const long BAGGAGE_PRESENT              = 122; /**< <b>Scope:</b> private */
    const long BAGGAGE_ABSENT               = 123; /**< <b>Scope:</b> private */
    const long BAGGAGE_OVERSIZED            = 124; /**< <b>Scope:</b> private + platform */
    const long BAGGAGE_ILLICIT_WEIGHT_CHANGE = 125; /**< <b>Scope:</b> private @note Used only for Conveyor-
not for new ConveyorSBD-Component, deprecated for CUSS 1.3 */
    const long BAGGAGE_READY_FOR_TAKE_IN    = 126; /**< <b>Scope:</b> public @note Used only for Conveyor- not
for new ConveyorSBD-Component, deprecated for CUSS 1.3 */
    const long BAGGAGE_TOO_MANY_BAGS        = 127; /**< <b>Scope:</b> private + platform */
    const long BAGGAGE_DELIVER              = 128; /**< <b>Scope:</b> private @note Used only for Conveyor-
not for new ConveyorSBD-Component, deprecated for CUSS 1.3 */
    const long BAGGAGE_UNEXPECTED_BAG       = 129; /**< <b>Scope:</b> private + platform */
    const long BAGGAGE_TOO_HIGH             = 130; /**< <b>Scope:</b> private + platform */
    const long BAGGAGE_TOO_LONG             = 131; /**< <b>Scope:</b> private + platform */
    const long BAGGAGE_TOO_FLAT             = 132; /**< <b>Scope:</b> private + platform  */
    const long BAGGAGE_TOO_SHORT            = 133; /**< <b>Scope:</b> private + platform  */
    const long BAGGAGE_PARKED               = 134; /**< <b>Scope:</b> private @note Used only for Conveyor-
not for new ConveyorSBD-Component, deprecated for CUSS 1.3 */
    const long BAGGAGE_INVALID_DATA         = 135; /**< <b>Scope:</b> private + platform */
    const long BAGGAGE_TRANSPORT_FAILED     = 136; /**< <b>Scope:</b> private + platform @note Used only for
Conveyor- not for new ConveyorSBD-Component, deprecated for CUSS 1.3 */
    const long BAGGAGE_WEIGHT_OUT_OF_RANGE  = 137; /**< <b>Scope:</b> private + platform */
    const long BAGGAGE_JAMMED               = 138; /**< <b>Scope:</b> private + platform  */
    const long BAGGAGE_EMERGENCY_STOP       = 139; /**< <b>Scope:</b> private + platform  */
    const long BAGGAGE_RESTLESS             = 140; /**< <b>Scope:</b> private + platform  */
    const long BAGGAGE_TRANSPORT_BUSY       = 144; /**< <b>Scope:</b> private + platform  */
    const long BAGGAGE_MISTRACKED           = 145; /**< <b>Scope:</b> private + platform  */
    const long BAGGAGE_UNEXPECTED_CHANGE    = 146; /**< <b>Scope:</b> private + platform  */
    const long BAGGAGE_ACCEPTED             = 147; /**< <b>Scope:</b> private + platform  */
    const long BAGGAGE_DELIVERED            = 148; /**< <b>Scope:</b> private + platform  */
    const long BAGGAGE_INTERFERENCE_USER    = 149; /**< <b>Scope:</b> private + platform  */
    const long BAGGAGE_INTRUSION_SAFETY     = 150; /**< <b>Scope:</b> private + platform  */

    const long FORMAT_ERROR               = 201; /**< <b>Scope:</b> private + platform (platform only for
output classes) */
    const long LENGTH_ERROR               = 202; /**< <b>Scope:</b> private + platform (platform only for
output classes) */
    const long DATA_MISSING               = 203; /**< <b>Scope:</b> private + platform (platform only for
output classes) */
    const long PHYSICAL_ERROR             = 204; /**< @note NOT used in CUSS 1.0 */
    const long DATA_PRESENT               = 205; /**< <b>Scope:</b> private */

    const long CONSUMABLES                = 301; /**< <b>Scope:</b> public */
    const long HARDWARE_ERROR             = 302; /**< <b>Scope:</b> public */
    const long CRITICAL_SOFTWARE_ERROR    = 303; /**< <b>Scope:</b> public */
    const long NOT_REACHABLE              = 304; /**< <b>Scope:</b> public */
    const long NOT_RESPONDING             = 305; /**< <b>Scope:</b> public */
    const long THRESHOLD_ERROR            = 306; /**< <b>Scope:</b> public */
    const long THRESHOLD_USAGE            = 307; /**< <b>Scope:</b> public */
    const long CONFIGURATION_ERROR        = 308; /**< <b>Scope:</b> public */
    const long SESSION_TIMEOUT            = 309; /**< <b>Scope:</b> private + platform (application related)
*/
    const long KILL_TIMEOUT               = 310; /**< <b>Scope:</b> private + platform (application related)
*/

    const long CUSS_MANAGER_REQUEST       = 801; /**< <b>Scope:</b> private + platform */
    const long SP_SYSTEM_MANAGER_REQUEST  = 802; /**< <b>Scope:</b> private + platform */
    const long AL_SYSTEM_MANAGER_REQUEST  = 803; /**< <b>Scope:</b> private + platform */
    const long CL_APPLICATION_REQUEST     = 804; /**< <b>Scope:</b> private + platform */
    const long AL_APPLICATION_REQUEST     = 805; /**< <b>Scope:</b> private + platform */

    // base definition for application generated events (technical)
    const long APPLICATION_TECHNICAL_FIRST = 400; /**< <b>Scope:</b> private */
    const long APPLICATION_TECHNICAL_LAST  = 499; /**< <b>Scope:</b> private */

    // base definition for application generated events (security)
    const long APPLICATION_SECURITY_FIRST  = 500; /**< <b>Scope:</b> private */
```

```
    const long APPLICATION_SECURITY_LAST   = 599; /**< <b>Scope:</b> private */

    // base definition for application generated events (business)
    const long APPLICATION_BUSINESS_FIRST  = 900; /**< <b>Scope:</b> private */
    const long APPLICATION_BUSINESS_LAST   = 999; /**< <b>Scope:</b> private */
};

#endif // CODES_IDL
```

characteristics.idl (Virtual component characteristics)

```
//------------------------------------------------------------------------------
//
//  File:      characteristics.idl
//
//  Purpose:   CUSS virtual components characteristics
//
//  Date:      17.06.2013
//
//  Version:   1.3
//
//  Author:    IATA Passenger Experience Management Group: CUWG CUSS-TSG
//
//  Copyright(c) 2003,2009,2013 International Air Transport Association, All Rights Reserved
//
//  Note:      Please refer to the CUSS 1.3 Technical Specification for more information
//
//------------------------------------------------------------------------------

#ifndef CHARACTERISTICS_IDL
#define CHARACTERISTICS_IDL

#pragma prefix "cuss.iata.org"

#include "codes.idl"
#include "types.idl"

/** Definition of the Virtual Component Characteristics
 *
 *  @note: Some attributes may not be applicable depending on the corresponding real component.
 *         Non-applicable values are either represented as:
 *         -1                    (for attributes of type long or string) or
 *         nonApplicableValue    (for attributes of enumerated types).
 */

module Characteristics
{

    /** Common Characteristic definition
        Manufacturer specifications for system management purposes but not restricted to it.
    */
    interface Manufacturer
    {

        /** Component identification for use by system manager.
            This identification gives a more precise definition of the component, e.g. ATB-PRINTER-BIN1
        */
        readonly attribute string realComponentIdentification;

        /** Describes whether the firmware can be updated or not */
        readonly attribute boolean downloadableFirmware;

        /** Version of firmware/software */
        readonly attribute string firmwareVersion;

        /** Name of manufacturer */
        readonly attribute string manufacturerName;

        /** Model number of hardware component */
        readonly attribute string modelNumber;

        /** Serial number of hardware component */
        readonly attribute string serialNumber;
    };

    /** Common Characteristic definition */
    interface MediaType
    {
        /** Definition of media types */
        enum MediaTypeDef
        {
          nonApplicableMediaType, /**< Non applicable characteristic value */

          MagneticStripe,         /**< Documents with a magnetic stripe */
          Chip,                   /**< Documents with a chip like RF-Baggage tags */
```

```
          Printed,               /**< Any printed document (OCR/BarCode/Plain paper) */
          JIS                    /**< JIS cards */
        };

        /** Attribute containing one media type */
        readonly attribute MediaTypeDef type;
    };

    /** Definition of the list of media types */
    typedef sequence <MediaType> MediaTypeListDef;

    /** Common Characteristic definition */
        interface MediaTypeList
    {
        /** List of media types */
        readonly attribute MediaTypeListDef mtList;
    };

    /** Common Characteristic definition */
    interface Location
    {

        /** Supported image types  */
        enum ImageType
        {
          nonApplicableImageType, /**< Non applicable characteristic value */

          notAvailable,          /**< no image available */
          BMP,                   /**< Microsoft bitmap file */
          JPEG,                  /**< JPEG file */
          PNG,                   /**< Portable network graphics file */
          Flash                  /**< Macromedia flash file */
        };

        /** URL to location image */
        readonly attribute string Map;

        /** The type of the image as specified above */
        readonly attribute ImageType mapType;

        /** URL to usage image/animation */
        readonly attribute string howTo;

        /** The type of the image/animation as specified above */
        readonly attribute ImageType howToType;

        /** Definition of where to find components */
        enum LocationType
        {
            nonApplicableLocationType, /**< Non applicable characteristic value */

            inKiosk,                  /**< device is located in the kiosk */
            inArea                    /**< device is located outside the kiosk */
        };

        /** Where to find the component */
        readonly attribute LocationType componentLocation;
    };

    /** Common Characteristic definition */
    interface ComponentFonts
    {
        /** CUSS supported barcodes */
        enum BarcodeStandard
        {
          nonApplicableBarcodeStandard, /**< Non applicable characteristic value */

          Code39,                      /**< Barcode definition */
          Code128,                     /**< Barcode definition */
          Code2of5                     /**< Barcode definition */
        };

        /** specifies which of the above standards was used */
        readonly attribute BarcodeStandard usedStandard;

        /** Specification of a single font */
        struct FontSpec
        {
          string         fontName;          /**< Name of the font */
          sequence<long> fontSizes;         /**< List of available font sizes, not set if font is a vector font
*/
```

```
        boolean        vectorFont;        /**< Font attribute */
        boolean        bold;              /**< Font attribute */
        boolean        italic;            /**< Font attribute */
        boolean        underlined;        /**< Font attribute */
        boolean        strikeThrough;     /**< Font attribute */
        boolean        reverse;           /**< Font attribute */
        boolean        superScript;       /**< Font attribute */
        boolean        subScript;         /**< Font attribute */
        long           colorDepth;        /**< Font attribute */
        long           spacing;           /**< Font attribute */
        long           characterLength;   /**< Font attribute (0 = variable length, n = fixed length) */
    };

    /** FontList declaration */
    typedef sequence<FontSpec> FontList;

    /** Fonts available from this component */
    readonly attribute FontList Fonts;
};

/** Common Characteristic definition */
interface Bin
{

    /** Describes the maximum number of documents a bin can hold */
    readonly attribute long BinSize;

    /** Shows the high threshold of the bin if corresponding sensor is installed
        Refers to the MEDIA_HIGH event in <i>codes.idl</i>
    */
    readonly attribute long AllmostFullLevel;

    /** Shows low threshold of the bin if corresponding sensor is installed
        Refers to the MEDIA_LOW event in <i>codes.idl</i>
    */
    readonly attribute long AllmostEmptyLevel;

    /** Shows the current number of documents in the bin.
     *  This value is adjusted by the platform automatically
     *  after documents have been printed.
     */
    readonly attribute long currentNoOfDocuments;
};

/** Common Characteristic definition */
interface IOMode
{
    /** MediaInput/MediaOutput supported modes */
    enum InputOutputMode
    {
      nonApplicableInputOutputMode, /**< Non applicable characteristic value */

      CheckIn,                      /**< Check-in mode for ATB printers */
      Revalidation                  /**< Revalidation mode for ATB printer */
    };

    /** The currently used mode for reading/writing. */
    readonly attribute InputOutputMode mode;

    /**
     *  Set the input/output mode for ATB printers.
     *
     *  @param  appRef  A valid application reference
     *  @param  mode    The input/output mode to be used (check-in or Revalidation)
     *
     */
    returncodes::rc setIOMode(in types::reference appRef, in InputOutputMode mode);
};

/** Capture characteristics */
interface Capture : Bin, Manufacturer { };

/** DataInput characteristics
 *
 *  @note  In CUSS 1.0, supportedDataTypes attribute is missing.
 *         The following data types are assumed:
 *         string         as clockDataType for Clock device
 *         const long     as switchDataType for sensor devices
 *
 */
interface DataInput : Manufacturer
```

```
{
    /** time difference in hours relative to GMT
     *
     * @note   This applies only to Clock device
     *
     */
    readonly attribute long timeZone;
};

/** DataOutput characteristics */
interface DataOutput : Manufacturer { };

/** Dispenser characteristics */
interface Dispenser : Bin, Location, Manufacturer
{
   /** Dispenser types */
   enum DispenserType
   {
       nonApplicableDispenserType, /**< Non applicable characteristic value */

       real_,                      /**< User can't access media without an <i>offer-command</i> */
       virtual_                    /**< User can access media all the time */
   };

   /** Specifies the kind of the dispenser */
   readonly attribute DispenserType kind;
};

/** Feeder characteristics */
interface Feeder : Bin, Manufacturer { };

/** Description of data types used by the MediaInput characteristics */
enum DataType
{
    nonApplicableDataType, /**< Non applicable characteristic value */

    AEA,                   /**< CUSS - AEA data type*/
    MSG,                   /**< CUSS - MSG data type*/
    SVG                    /**< CUSS - SVG data type*/
};

/** Definition of the list of data types */
typedef sequence <DataType> DataTypeList;

/** MediaInput characteristics */
interface MediaInput : IOMode, MediaTypeList, ComponentFonts, Location, Manufacturer
{
    /** Describes the type of media reader */
    enum ReaderType
    {
        nonApplicableReaderType, /**< Non applicable characteristic value */

        Motorized,               /**< Motorized reader like standard card readers */
        DIP,                     /**< Manual insertion or removal of documents */
        Swipe,                   /**< Document has to be swiped through reader manually */
        Contactless,             /**< Document is read via an antenna (radio frequency) */
        FlatbedScan,             /**< Standard scanner technology or camera */
        PenScan                  /**< Standard barcode reader technology */
    };

    /** The kind of reader which is handled by this component */
    readonly attribute ReaderType typeOfReader;

    /** The list of data types supported by this component */
    readonly attribute DataTypeList supportedDataTypes;

    /** Describes the type of data stream that is supported by this component.
*
*  @note   This attribute is not used in CUSS 1.0
*
     */
    readonly attribute DataType setupDataType;

    /** The number of tracks that can be read by the components.
        \li 1 indicates only track 1 is readable
        \li 2 indicates track 1 and track 2 are readable
        \li 3 indicates tracks 1 to 3 are readable, and so on
    */
    readonly attribute long numberOfTracks;
};
```

```
/** MediaOutput characteristics */
interface MediaOutput : IOMode, MediaTypeList, ComponentFonts, Location, Manufacturer
{
    /** Media types */
    enum MediaType
    {
      nonApplicableMediaType, /**< Non applicable characteristic value */

      Ticket,                /**< TAT- or ATB ticket */
      BoardingPass,          /**< Boarding pass */
      GeneralPurposeDoc,     /**< General purpose document */
      BaggageTag,            /**< Baggage tag */
      InsertedDoc,           /**< Document inserted by the user */
      Card                   /**< Any type of card */        };

    /** Attribute containing the media type */
    readonly attribute MediaType type;

    /** The list of data types supported by this component */
    readonly attribute DataTypeList supportedDataTypes;

    /** Size of the internal data buffer */
    readonly attribute long bufferSize;

    /** The number of tracks that can be written by the components.
        \li 1 indicates only track 1 is writable
        \li 2 indicates track 1 and track 2 are writable
        \li 3 indicates tracks 1 to 3 are writable, and so on
    */
    readonly attribute long numberOfTracks;

    /** The minimum length of a document measured in MilliMeters */
    readonly attribute long minDocumentLength;

    /** The maximum length of a document measured in MilliMeters */
    readonly attribute long maxDocumentLength;

    /** Printing technology specification
     *
     *  @note   attribute for mediaTransferType is missing in CUSS 1.0 IDL.
     *          Instead, its value (DirectThermal or ThermalTransfer )
     *          should be inserted somewhere inside the string representing
     *          Manufacturer::ModelNumber attribute. This was agreed upon
     *          in order to keep CUSS 1.0 IDLs backward-compatible to 0.2.1.
     *
     */
    enum MediaTransferType
    {
      nonApplicableMediaTransferType, /**< Non applicable characteristic value */

      DirectThermal,                  /**< Thermal printing device */
      ThermalTransfer                 /**< Ribbon printing device */
    };


    /** The maximum printing size in X direction measured in MilliMeters */
    readonly attribute long maxPrintSizeX;

    /** The maximum printing size in Y direction measured in MilliMeters */
    readonly attribute long maxPrintSizeY;

    /** Printing orientations */
    enum PrintOrientationDef
    {
      nonApplicablePrintOrientation, /**< Non applicable characteristic value */

      Portrait,                     /**< printing orientation */
      Landscape                     /**< printing orientation */
    };

    /** The current print orientation. */
    readonly attribute PrintOrientationDef printOrientation;

    /**
     *  Sets the printing orientation to be used by this component.
     *
     *  @param  appRef      A valid application reference
     *  @param  orientation The printing orientation (Portrait or Landscape)
     *
     */
    returncodes::rc setPrintOrientation(in types::reference appRef, in PrintOrientationDef orientation);
```

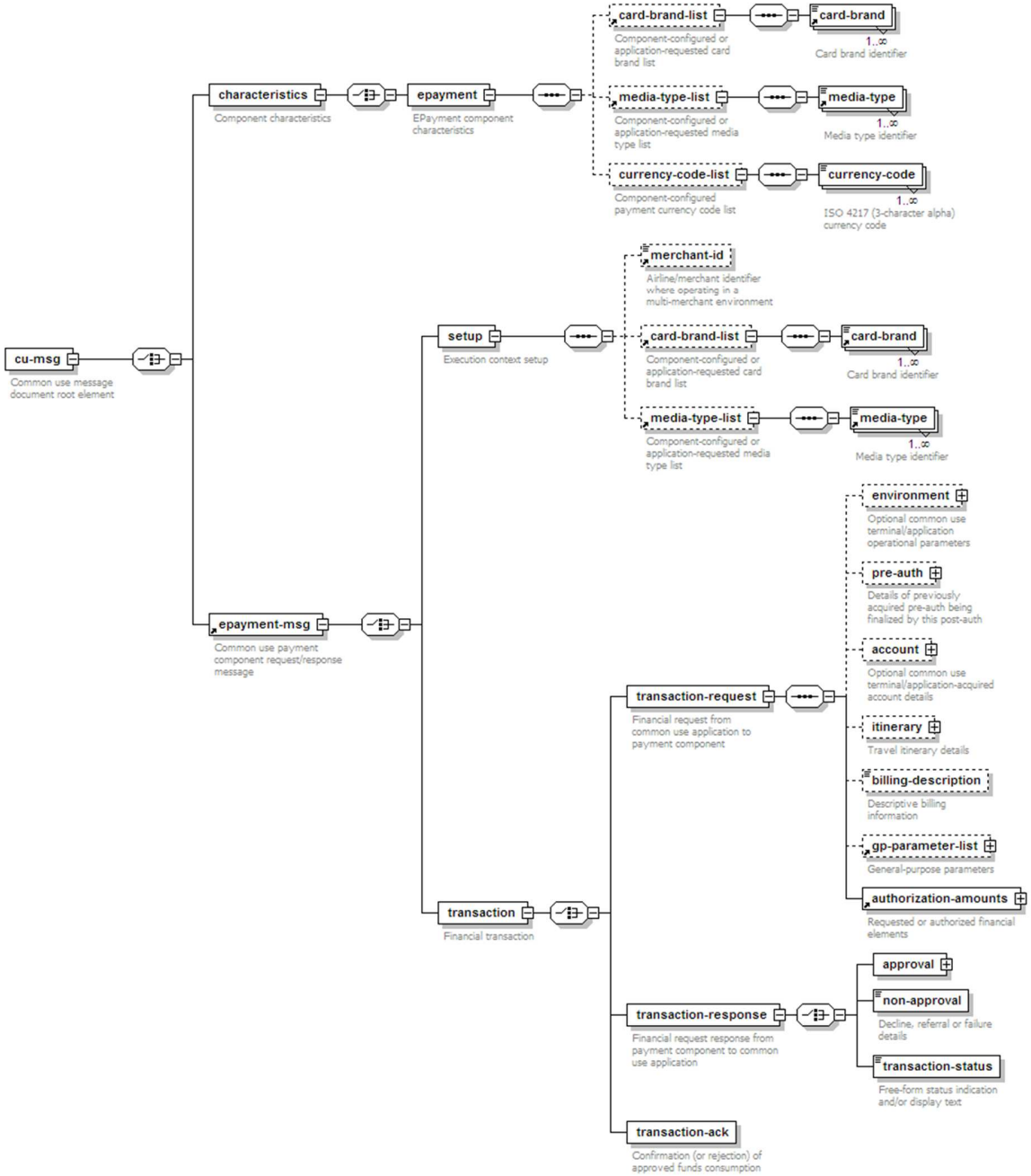
```
};

/** Storage characteristics */
interface Storage : Manufacturer
{

    /** Specifies the total size available for an application on a disk */
    readonly attribute long size;

    /** Specifies the path to writeable/readable location
        (all path specifications end with a separator, e.g. slash or backslash)
    */
    readonly attribute string path;
};

/** Display characteristics */
interface Display : Location, Manufacturer
{
  /** Resolution list definition */
  typedef sequence<long> ResolutionList;

  /** List of supported screen resolutions
      \li  800 indicates a resolution of  800 by  600
      \li 1024 indicates a resolution of 1024 by  768
      \li 1280 indicates a resolution of 1280 by 1024
      \li 1600 indicates a resolution of 1600 by 1200

      @attention Currently only one screen resolution should be used if the
                 touch screen overlay is not automatically re-calibrated
  */
  readonly attribute ResolutionList displayResolution;

  /** Currently used screen resolution. */
  readonly attribute long currentResolution;

  /**
   *  Sets a new resolution for the display.
   *
   *  @param  appRef     A valid application reference
   *  @param  resolution The screen resolution to be used by the application
   *
   */
  returncodes::rc setScreenResolution(in types::reference appRef, in long resolution);

  /** Physical screen size measured in MilliMeter */
  readonly attribute long screenDiagonal;
};

/** UserInput characteristics */
interface UserInput : Location, Manufacturer { };

/** UserOutput characteristics */
interface UserOutput : Location, Manufacturer { };

/** Network characteristics
    Standard attributes only for this component.
*/
interface Network : Manufacturer { };
/** Application characteristics */
interface Application : Manufacturer
{
    /** Kiosk Application identification */
    readonly attribute types::akID identification;

    /** Definition of the different address types */
    enum ContactAddressType
    {
        nonApplicableContactAddress, /**< Non applicable characteristic value */

        None,                        /**< No address available */
        Pager,                       /**< Page address */
        EMail,                       /**< EMail address */
        Phone,                       /**< Phone address */
        Postal,                      /**< Postal address */
        Fax,                         /**< Fax address */
        Network,                     /**< Network address (Remote network support application) */
        URL,                         /**< Uniform Resource Locator (Internet support application) */
        IOR                          /**< Interoperable Object Reference (CORBA support application) */
    };

    /** Definition for a single contact field */
```

```
    struct Contact
    {
      string             name;    /**< Name of the person to contact */
      string             company; /**< Name of the company to contact */
      string             note;    /**< Unspecific note on person or company */
      string             address; /**< Address to be used */
      ContactAddressType type;    /**< Specifies the type of the address */
    };

    /** Definition for the contact list */
    typedef sequence <Contact> ContactList;

    /** The list of all available contacts */
    readonly attribute ContactList allContacts;

    /** Specifies the first of IP-Port range that can be used by this application */
    readonly attribute long firstIPPort;

    /** Specifies the last of IP-Port range that can be used by this application */
    readonly attribute long lastIPPort;
};

/** Scale characteristics */

interface BaggageScale
{
    /** The maximum weight of the baggage (in grams) */
    readonly attribute long maxWeight;
};

/** ConveyorSBD characteristics
*  @note This is the new definition for CUSS 1.3. Older definitions are deprecated
*  from CUSS version 1.3 and will be completely removed in CUSS 1.5 latest.
 */

interface ConveyorSBD : Location, Manufacturer
{
    /** The maximum weight of the baggage (in grams) */
    readonly attribute long maxWeight;

    /** The maximum width of baggage (in millimeters) */
    readonly attribute long maxWidth;

    /** The maximum height of baggage (in millimeters) */
    readonly attribute long maxHeight;

    /** The maximum length of baggage (in millimeters) */
    readonly attribute long maxLength;

    /** The maximum number of bags a conveyor can handle */
    readonly attribute long maxBags;

  /** If true, conveyor has a security barrier (for user safety) */
  readonly attribute boolean barrierCapable;

  /** If true, conveyor system can detect intrusions at the front/user side (insertion) */
  readonly attribute boolean userInterferenceCapable;

  /** If true, conveyor system can detect intrusions behind the front/user side (verification/parking) */
  readonly attribute boolean safetyIntrusionCapable;
};

/** Conveyor characteristics
*  @note This interface definition is deprecated from CUSS 1.3. Application-suppliers
* are encouraged to implement the ConveyorSBD characteristics interface
* for self-service baggage check-in support.
*/

interface Conveyor : Location, Manufacturer
{
    /** Conveyor types */
    enum ConveyorType
    {
        nonApplicableConveyorType,  /**< Non applicable characteristic value */

        Scale,                      /**< Conveyor has a scale only */
        BCR,                        /**< Conveyor has a barcode reader only */
        Scale_BCR,                  /**< Conveyor has a scale and a barcode reader */
        Scale_ParkPosition,         /**< Conveyor has a scale and a park position for bags */
        BCR_ParkPosition,           /**< Conveyor has a barcode reader and a park position for bags */
```

```
            Scale_BCR_ParkPosition      /**< Conveyor has a scale, a barcode reader and a park position for
bags */
        };

        /** the maximum weight of the baggage */
        readonly attribute long maxWeight;

        /** Specifies the kind of the conveyor */
        readonly attribute ConveyorType typeOfConveyor;

        /** the maximum width of baggage */
        readonly attribute long maxWidth;

        /** the maximum height of baggage */
        readonly attribute long maxHeigth;

        /** the maximum length of baggage */
        readonly attribute long maxLength;

        /** the guarenteed number of bags that can be stored at least in the parking position */
        readonly attribute long guarenteedNoOfBags;

        /** the actual number of bags in the parking position */
        readonly attribute long currentNoOfBagsParked;
    };


};

#endif // CHARACTERISTICS
```

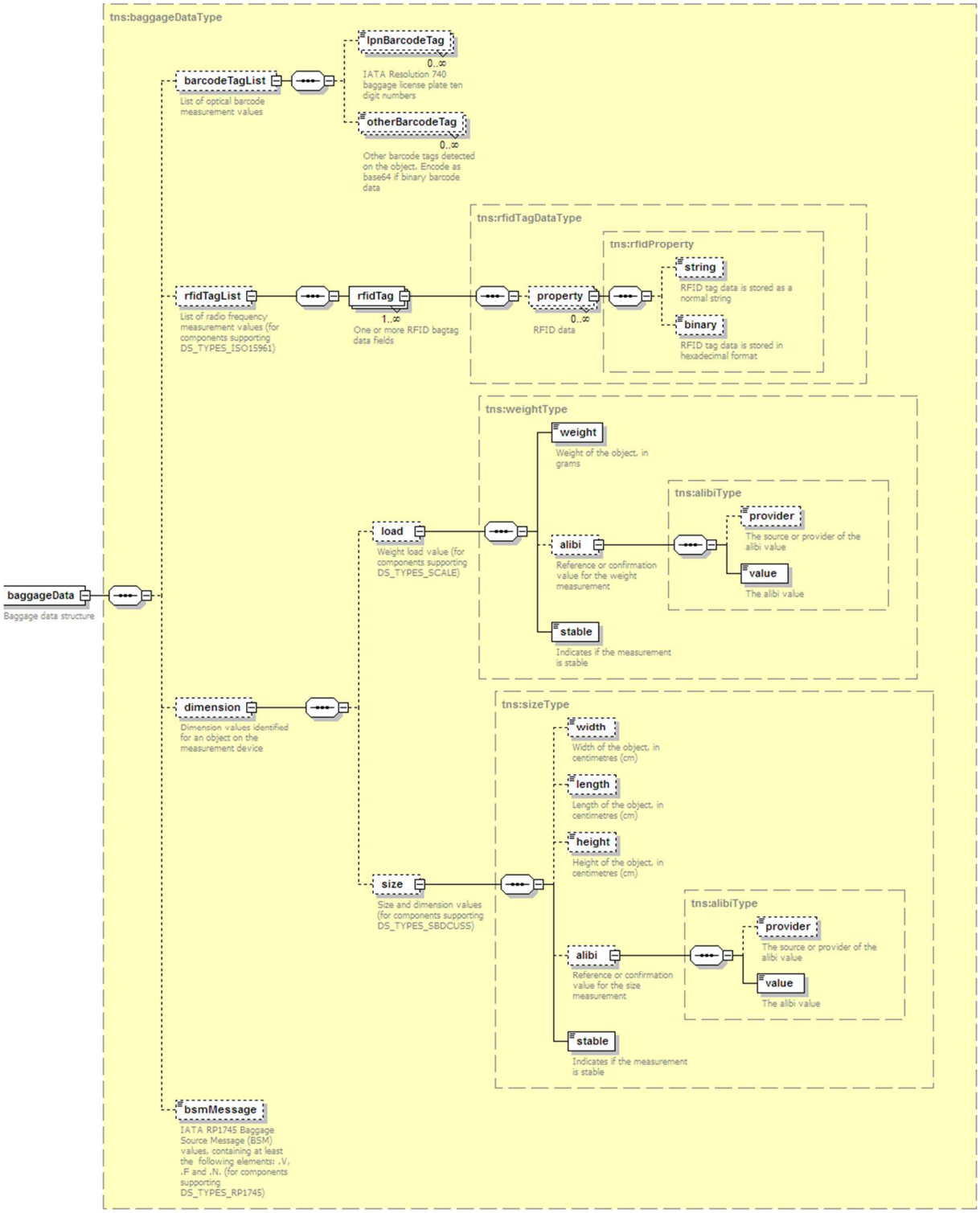
CUSS.PAYMENT.XSD (Generic Payment XML messages)

Please refer to Section 7.19 for sample messages created with this schema definition.



CUSS.SBD.XSD (Scales and Self Bag Drop)

Please refer to Section 7.17 and 7.18 for sample messages created with this schema definition.



Appx D: AEA Printer Standard and Usage

CUSS mandates that a kiosk and CUSS platform be equipped to printer AEA printer format Boarding Pass documents (See Section 4.1.1.) For more information please refer to the following AEA standards documents available from http://www.aea.be:

- Automated Ticket & Boarding Pass (ATB) Technical Specifications (2009)
- Parametric Baggage Tag Data Concept (2002)
- Self Service Specifications (August 2001)
- Self Bag Drop Specifications (AEA2012-2 March 2013)
- IATA Resolution 792 (BCBP)

AEA printers must support all the requirements of IATA Resolution 792 for bar coded boarding passes, including a minimum print resolution of 200dpi. A CUSS platform may use any AEA-compliant printer implementation (emulated in software or via printer firmware) to print documents according to the specification.

The CUSS Technical Specification also uses the AEA standard for communicating with Self Bag Drop (SBD) devices. This usage is covered under Chapter 7.16 *Integrated Baggage Conveyors.* This Appendix only covers AEA as used for printing.

Version of AEA Printer Specification Supported

CUSS 1.3 platforms must support AEA2009 or later for ATB printers and bagtag printers.

CUSS 1.2 platforms must support AEA2008 for ATB and bagtag printers. For CUSS 1.0 and CUSS 1.1 compliant kiosks, the platform must support the AEA99 version of the specification.

The platform must also implement the AEA Self Service Specifications (resource management via context switching) to manage multiple applications, regardless of the version of the AEA specification.

The CUSS platform may optionally support a newer version of the standard (such as AEA2010 or later) if this newer version is backwards compatible with the minimum AEA versions indicated above, and continues to meet the additional guidelines listed below.

To report which version of the AEA standard is available for use by applications, the platform must include the version as part of the firmwareVersion characteristic of all components of the AEA printer device (bagtag or ATB) with the following syntax (do not include comments in brackets):

- AEA2009 [or later version required for CUSS 1.3 printers]
- AEA2008 [or later version required for CUSS 1.2 printers]

- AEA1999
- AEA2001
- AEA2002 [for previous versions of CUSS]

The native AV command in AEA must also be supported by the platform and must report the version of AEA supported by the printer components. For example, the AV command would return IERAVOK09#ATB in accordance with the AEA specification, indicating AEA2009 support.

If an application attempts to use a feature of a newer AEA version that is not reported as being available by the platform in this fashion, the application may receive AEA errors when attempting to load or print AEA streams.

PCX Logo Format Specification

This section is based on CUSS 1.0 Addendum A.1.8 and A.1.20.

The AEA standard does not clearly define the logo format to be used by AEA printers. All CUSS platforms and applications must only use PCX format logos that abide by the following guidelines:

1. Logo source resolution is 200dpi (8d/mm.) It is up to the CUSS platform provider to convert the logo data appropriately to appear properly on the printer their platform uses. This resolution is based on existing AEA printers and common ticket printers. This resolution is crisp enough to provide suitable output even on higher-resolution GPP printers providing AEA emulation.

2. PCX format must be black/white, 1bpp, where a "1" bit indicates a black dot.

3. The specific format of the LT stream can be either "LTxxyyyy<HEX-encoded PCX data>" or "LTxxyyyy<binary PCX data>". For example, since the PCX header starts 0x0A000101 (more or less), the LT stream would start, for logo 42, as "LT42yyyy0A000101..." for hex mode or as "LT42yyyy<0x0A><0x00><0x01><0x01>..." for binary mode. CUSS applications can use either mode, and it is up to the platform provider to convert as needed for the format expected by their physical printer (or AEA emulation.)

4. The yyyy length value indicates the length of the LT stream data. ie if your PCX files is 2300 bytes, then the text encoding is 4600 bytes, and the stream would be "LT4246000A0000101...". If your PCX file is binary and is 4700 bytes, the stream would be “LT4700<0x0a><0x00>…”

5. The maximum PCX logo file size is 4999 bytes if hex mode is used, and 9999 bytes for binary mode.

The PCX file format supports five separate header format versions, all of which support monochrome files as described above. When monochrome data is used, the image data encoding is consistent across all versions. The PCX versions are:

```
0=PC Paintbrush v2.5
2=PC Paintbrush v2.8 w palette information
3=PC Paintbrush v2.8 w/o palette information
4=PC Paintbrush/Windows
5=PC Paintbrush v3.0+
```

PCX version 0 supports all requirements of CUSS/AEA. Later versions introduce support for 256+ colors, custom palettes, and other features that are not used in CUSS.

It is up to the platform provider to ensure that valid monochrome PCX logo data loads and prints on the printer used in their kiosk, regardless of what limitations exist within that printer with regards to PCX header/version support. In other words, it is up to the CUSS platform to adjust or "normalize" the PCX data it receives from application so that it does not cause problems on the printers that the platform uses to print AEA documents and bag tags.

As an example of a potential problem that could occur, some PCX image versions include a 256-color palette after image data. This extra data is not technically needed for monochrome images, but it is compliant with the PCX standard may be included in data sent by some application. A platform provider would need to strip this extra data if, for example, that data caused problems on their printer. Here are some additional examples of adjustments the platform might need to make to accommodate the limitations of their AEA printer:

- Add/Remove 769-byte palette at end of data
- Reset internal header version if a specific value is required by printer.
- Adjust coordinate margins / resolution in header if needed by printer
- Set header 16-color palette values if specific values needed by printer
- Reject as invalid other PCX data that is *NOT* monochrome

Please consult the PCX file format specification for more information.

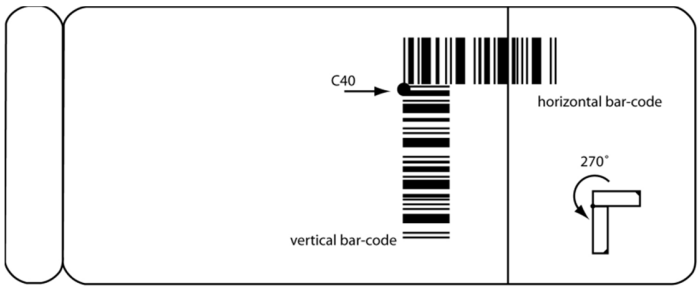
Barcode Orientation

This section is based on CUSS 1.0 Addendum A.1.21 and the AEA2007 specification.

The AEA99 specification merely makes the following comment: "Note for orientation: The lower left corner of the barcode is equal to the lower left corner of the position matrix e.g. C05."

For horizontal barcodes, this is clear and indicates that the rectangular barcode printout extends up and to the right of the element position.

For vertical barcodes, the only mention is of a 270 degree rotation. The rotation guideline for template Text elements specifies that this is a counterclockwise rotate around the lower left corner of the barcode. As such, the printed vertical barcode extends down and to the right of the element position, as demonstrated by the following approximate diagram for position C40:



8" ATB document (with binding stub)

Horizontal barcodes are AEA barcode types 0-9, and vertical barcodes are barcode types Q-Z.

PDF417 2D Barcode Printing

This section is based on CUSS 1.0 Addendum A.1.9 and the AEA specification.

A CUSS 1.3 platform must support AEA2009, and it must support the PDF417 2D barcode printing capability of AEA2009.

PDF417 barcodes were first introduced in AEA2002, but this support was poorly defined. To clarify this original definition, CUSS 1.0 listed the specific requirements and syntax for printing PDF417 2D barcodes. This CUSS 1.0 clarification was then included as part of the AEA2007 specification and later versions, including AEA2009 as require by CUSS 1.3.

For clarity, the description of PDF417 printing support taken from CUSS 1.0 and from AEA2007/AEA2008 is included here:

The required implementation on CUSS platforms is:

1. **BARCODE POSITION** - As defined in AEA, the coordinate is a standard row/column location. In horizontal mode, the bottom left corner of the barcode coincides with the bottom left corner of the character at that row/column. In vertical mode, the upper left corner of the barcode is the bottom left corner of the row/column, i.e. the barcode is underneath the row completely.

2. **BARCODE TYPE** - As defined in AEA, this is "6" for a horizontal barcode and "R" for a vertical barcode.

3. **BARCODE HEIGHT** - As defined in AEA, the height of the horizontal barcode, in millimeters (1-99.) Because some vendors may not use PDF417 libraries that support exact height specifications, the actual barcode height should be as close as reasonably possible to the requested height.

4. **BARCODE MAGNIFICATION** (1-9) - This is the more ambiguous aspect of a 2D barcode. This magnification parameter affects the amount of error-correcting codes (ECC) overhead included within the barcode. A higher value should result in a wider barcode with more redundancy. For PDF417, the following values are used:

1 == PDF417 ECC LEVEL 0
2 == PDF417 ECC LEVEL 1
3 == PDF417 ECC LEVEL 2
4 == PDF417 ECC LEVEL 3
5 == PDF417 ECC LEVEL 4
6 == PDF417 ECC LEVEL 5
7 == PDF417 ECC LEVEL 6
8 == PDF417 ECC LEVEL 7
9 == PDF417 ECC LEVEL 8

5. **BARCODE RATIO** (1-3) - This indicates the size of the elements within the 2D barcode, which indicates a “fine”, “regular” or “coarse” 2D barcode appearance, with respect to the height and width of the codeword modules. Here is a recommended approximation for a 200dpi printer with the standard 3:1 PDF417 module ratio:

1 – “fine” – rows that are 0.75mm high (6 pixels)
2 – “regular” – rows that are 1.125mm high (9 pixels)
3 – “coarse” – rows that are 1.5mm high (12 pixels)

Please refer to the AEA2009 specification for information on printing additional 2D barcodes types such as QRCode or Aztec.

Barcode128 subtypes 128A, 128B, 128C

This section is based on CUSS 1.0 Addendum A.1.25.

There is lack of clarity in the AEA specification regarding Code 128 implementation on AEA documents. The required implementation on CUSS platforms is as follows:

- If the barcode data is all numeric, print as Code 128C.
- If the barcode data also contains lowercase characters, print as Code 128B
- In all other cases, print as Code 128A.

Please note that CUSS 1.3 uses AEA2009. The AEA barcode identifiers “4” and “V”, which in previous versions of AEA referred to Barcode 128 without check digit, are no longer available (they now refer to new DataMatrix and Aztec 2D barcodes.) Applications that previously used 4 and V must now use 7 and W and include the correct Barcode 128 check digit.

Multi-document AEA print streams

This section is based on CUSS 1.0 Addendum A.1.39.

The ability to share data and print multiple documents within a single AEA data stream is a valid and useful feature of the AEA printer standard. For example, baggage tags are often printed in sequences of consecutive tag numbers.

A CUSS kiosk supporting the AEA standard (AEA2008 required for CUSS 1.2) must support the multi-document printing capability of the AEA standard, and must generate properly-formed AEA responses to such requests. A compliant CUSS platform must not return FORMAT_ERROR in response to a valid AEA stream that produces multiple documents

Printing multiple coupons from separate stock types (Feeders) however, is not permitted within a single directive: if printing multiple coupons then only single stock type (such as BoardingPass) is permitted. An application that wants to use this feature of AEA should also meet these conditions:

- Verify that the bin size of the Dispenser (see Characteristics) is sufficient to hold all the documents contained in the AEA stream before issuing the send() directive.
- Parse the AEA response string from the platform to determine which documents were actually printed.
- If the Dispenser cannot hold all the documents, first prompt the user to remove existing documents before issuing the new print request.
- If the Dispenser cannot hold all the documents even when empty (based on Bin characteristics), the application must break up the AEA stream into separate requests.
- Verify that any timeouts given in the send() requests are long enough to allow multiple documents to print.

Even if multiple coupons are printed in a single print request, the CUSS component will generate only one AEA status completion message in the component event.

Extended code page language support for AEA print streams

The AEA2008 and AEA2009 specifications allow an AEA printer to *optionally* support extended code pages for printing documents including alternate languages and characters.

Air carriers that wish to use extended language support should ensure that the documents produced at CUSS kiosks comply with any applicable IATA Resolution, Recommended Practice, and local airline/airport as well as interline partner requirements.

Extended language/codepage printing would usually be a requirement for a certain markets that would require local language support, such as domestic travel within many countries (such as Russia, China, Japan, etc.)

To print an AEA document in an extended language, a CUSS application must:

- Request the language using the setup() command sending an "EP#FONT=x" request
- Interpret the platform response to the EP request to see if the request was successful
- If successful, send the print data encoded in the requested codepage
- The application can query the current font using the "ES" request

Extended language printing is an *optional* component of the AEA print language: CUSS-compliant *kiosks are not required to support extended language printing* and printers and platforms can be compliant with AEA2009 even if they do not support printing in extended code pages.

If the platform is able to support the extended code page requests, it must use the following codepages (taken from the AEA2009 specification):

EP#FONT=**L**	Latin **(CP-850)**
EP#FONT=**R**	Cyrillic **(Windows-1251)**
EP#FONT=**C**	Chinese **(Windows-936)**
EP#FONT=**K**	Korean **(Windows-949)**
EP#FONT=**J**	Japanese **(Windows-932)**
EP#FONT=**A**	Arabic **(Windows-1256)**

If a CUSS application requests an extended language that the platform or printer does not support, then the platform shall provide the correct ERRE error response to the application.

An application must then implement the desired business logic to handle the case where it is operating on a CUSS kiosk that does not support printing documents in the language the application requested.

If an application does not specify a font using the ES command, or specifies a font that is not supported, the printer shall use the default codepage for printing the document. The AEA

specification does not specify a default codepage, so CUSS applications cannot assume a single default codepage exists and is consistent across all kiosk printers.

Here are additional comments regarding extended codepage printing in AEA2009:

- When the application does not issue an EP command to set the codepage, the *default* codepage used will vary from printer to printer, as this default behaviour is not specified in the AEA standard. Hence CUSS application must not expect a a consistent, default codepage across all CUSS kiosks.For example, these different codepages are used:

    a. IER640 ATBSSB134: **CP-437**
    b. IER567 96629-1.70: **CP-850**
    c. IBM CUSS GPP AEA Emulation: **Windows-1252**

- AEA documents are printed with left-to-right printing, even of the codepage language is right-to-left. A future version of AEA may clarify or alter this behavior.
- The field length indicator in the pectab refers to the byte length of the print stream request for that field. It does not refer to the length of the printed output. This is an important consideration when printing double-width characters, or multi-byte code pages.
- The extended codepage print also applies to the contents of templates.

Here is an example documented printed with the Cyrillic codepage, and its equivalent CP data stream as represented in Windows-1252:

```
                etix etkt etix etkt
ELECTRONIC TICKET   220 2134274420
                                * *
LUFTHANSA                      *
                              *
Ковалёв/Дмитрий MR            *
  MUC   LH 064  C 29SEP        *
  HAM                            * *

  3C
              LH  064 /002      RETURN

SEN
```

```
бизнес          002


Ковалёв/Дмитрий MR
   ETKT 220 2134274420
   MUC
   HAM
LUFTHANSA        SEN

LH 064  C 29SEP


G12    2100 3C
            Некурящи


00
```

```
CP#A#01W#CP#C01#02LUFTHANSA   #04Êîâàë¸â/Äìèòðèé MR#07/Ðîëüô MR   #08LH#09SEN
#0DMUC#0FRETURN#10KBR78#11MUC#14HAM#15SEN #17HAM#1ALH#1CX#1D064 #21C#22C#2429SEP#27G12
#2A2100#2D3C #2F3C #41Íåêóðÿùèõ#4BC#4C002 #4D002  #4FLH  064 /002  #71áèçíåñ#860#8D3#B6
#C1*#C300#C400#C7ETKT 220 2134274420#C8etix etkt etix etkt     #C9ELECTRONIC TICKET   220
2134274420#F1272#FA1 12111 1   #FF220#H8M1BRUHN/OLAF MR       EZNFODI MUCHAMLH 0064 272C3C
002  3010#
```

Additional language examples will be provided as made available to the TSG-CUSS technical group.

Restrictions on AEA Commands

The CUSS standard limits which AEA commands an application is permitted to send to the printer, in order to maintain the state of the print for all applications. For this reason, only the following commands are permitted:

Printer devices (AEA2009 or later):

setup() directive (see Section 3.6.6 for details):

CT, PT, PC, PS, LT, LC, LS, FT, FC, FS, FA, FR, TT, TC, TA, AV, ZS, PV, RI, RC
ES, EP
BT (without parameters)
BTT (bag tag printing only)

send() directive (see Section 3.6.8.2 for details):

CP, CI, TK, TI, TR
BTP (bag tag printing only)

Self Bag Drop (AEA2012-2 or later):

There are no restrictions on AEA commands sent to Self Bag Drop (SBD) devices.

Appx E: Technologies and Standards

Introduction

A CUSS Platform must support all of the following presentation technologies and standards. These technologies have been selected to allow a single application to run on multiple standard and commercially available browser technologies such as Flash and Silverlight, and within the standard Java environment. This allows predicatable and consistent behaviour for application providers.

Here is a summary list of what a CUSS 1.3 kiosk must provide, and the CUSS application suppliers can depend on when developing CUSS 1.3 applications.

- The standard CUSS browser is **Microsoft Internet Explorer 8 (IE8), as defined below**
- The standard CUSS java is **Oracle JRE7** (7u21 or later) for command line and browser plugin environments, as defined below

- **Adobe Flash 11** (version 11.7.7 or later) is available
- **Adobe Shockwave 12** (version 12.0 or later) is available
- **Adobe AIR 3** (version 3.7 or later) is available
- **Apple Quicktime 7** (version 7.7.3 or later) is available
- **Windows Media Player 11** (vesion 11.0 or later) is available
- **Microsoft Silverlight 5** (release 5.1 or later) is available
- **Adobe PDF Reader XI** (version 11.0 or later) is available
- **AEA2009** is available for document printing (ATB and BTP)
- **AEA2012-2** is available for Self Bag Drop devices
- **Google Chrome 27** is available in addition to the standard CUSS browser
- **Google Chrome Frame 21** is available within the standard CUSS browser

Interim Changes to the CUSS Technologies and Standards List

Generally speaking, the list above includes the latest versions available from the tool publishers that retain support for Windows XP, as list at the time of the final draft of an updated CUSS Technical Specification.

The CUSS standard is published infrequently compared to the pace of change in application development technologies and standards. To ensure that the CUSS Technical Specification can be kept up to date with new industry trends or with external requirements such as security guidance, the versions and technologies in this Appendix may be amended after publication.

To amend this appendix, the IATA Passenger Experience Management Group (PEMG) may publish the following document.

CUSS Technical Specification 1.3: Errata and Technologies Updates

When and if published, that document will supercede the content of this Appendix subject to the wording and content of the Document

Software Licensing and Distribution

This CUSS Standard requires various Tools and Technologies be available as part of a CUSS compliant environment. These tools and technologies come from 3rd parties, such as Microsoft, Oracle, Adobe, and Google.

Each external technology is subject to its provider's requirements regarding commercial licensing, redistribution, end user license agreements, export restrictions, security and data collection and data privacy policies, updating and patching, liability waivers and disclaimers, warranties, and other legal and commercial requirements for use.

Even though these tools are required as a component of a compliant CUSS kiosk, IATA and this CUSS Technical Standard do not abrogate or replace any of these 3rd party usage requirements.

It is up to each platform provider, platform operator, and application provider to review, understand, and comply with any such requirements as applicable to installing, deploying, and using 3rd party technologies at a specific site.

The Standard CUSS Java Environment

To ensure compatibility and portability from platform to platform and kiosk to kiosk, previous versions of the CUSS standard restricted and enforced that only certain, specific versions of the Sun/Oracle Java environment be available to applications.

This approach, which mandated specific versions 1.3.1_06 and 1.5.0_04 of the Java Runtime Environment, meant that CUSS kiosks were unable to keep Java up to date with security patches, time zone changes, and other ongoing fixes critical to the java environment. With release 1.3 of the CUSS Technical Standard, the java requirements are changing significantly:

CUSS 1.3 defines a Standard CUSS Java Environment. This standard java is Oracle JRE7, which must run as the default java environment for the java plugin within the standard CUSS browser, and from the system command line environment. The move to JRE7 provides these benefits:

- Uses a version of Java that is supported and updated by the vendor
- Allows CUSS sites to keep Java up to date with security fixes
- Operates correctly in modern systems like IE8 and Windows 7
- Provides backwards compatibility for applications that are designed for earlier Java versions
- Provides new features for application developers deploying into CUSS 1.3 environments

To be compliant with CUSS 1.3, kiosks **must** comply with the following requirements:

Java Runtime Environment (Command Line)

1. The kiosk must provide a command line java runtime environment

2. The default java available from the kiosk command line must be JRE7. The “default version” is considered the version that is run when “java.exe” is invoked on the kiosk without a specific path.

3. The version of JRE7 on the command line must be JRE 7u21 or later. Newer major releases (such as JRE8) must not be installed. New minor releases of JRE7 are permitted.

4. CUSS 1.3 kiosks are not required to provide earlier versions of the command line java listed in CUSS 1.0, 1.1 and 1.2.

5. If earlier versions of the command line java runtime environment are provided, they must be of the versions in this list:

    a. Java Runtime 1.3.1_06 (CUSS 1.0)
    b. Java Runtime 1.3.1_19 (CUSS 1.1)

c. Java Runtime 1.3.1_20 (CUSS 1.2)
d. Java Runtime 1.5.0_04 (CUSS 1.2)
e. Java Runtime JRE6u12 (CUSS 1.2)

6. If earlier versions of the command line java runtime are provided, they must be invoked by specifying the full path name of that version's java.exe file.

Java Runtime Environment (Browser Plugin)

1. The kiosk must provide a java runtime environment browser plugin and the default system browser (IE8) must be configured to use that browser plugin.

2. The default java available from the browser plugin must be JRE7. The "default version" is considered the version that is run when the browser plugin object is invoked with class ID "clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"

3. The version of JRE7 in the browser must be JRE 7u21 or later. Newer major releases (such as JRE8) must not be installed. New minor releases of JRE7 are permitted.

4. The JRE7 browser plugin must be the only plugin installed on the system. The earlier versions required for CUSS 1.0, 1.1 and 1.2 must not be present and must be uninstalled.

5. The JRE7 plugin must run properly if the application loads the browser plugin with any of the following class IDs (representing the versions used in CUSS 1.0-1.2.)
   a. clsid:CAFEEFAC-0013-0001-0006-ABCDEFFEDCBA
   b. clsid:CAFEEFAC-0013-0001-0019-ABCDEFFEDCBA
   c. clsid:CAFEEFAC-0013-0001-0020-ABCDEFFEDCBA
   d. clsid:CAFEEFAC-0015-0000-0004-ABCDEFFEDCBA
   e. clsid:8AD9C840-044E-11D1-B3E9-00805F499D93

6. The JRE7 browser plugin must be configured to avoid any popup messages or confirmations when the browser page is invoking any of the above class IDs.

Java Runtime Environment (General Comments)

A CUSS 1.3 platform provider must not accommodate any request from a CUSS application provider to install or use any version of java other than the JER7 command line and browser plug in environments described here.

A CUSS application that does not operate correctly with the java environments described above is not a CUSS-1.3-compliant application.

The Standard CUSS Browser Environment

In previous versions of CUSS 1.0, 1.1 and 1.2, the CUSS Technical Specification did not list a particular browser environment requirement, allowing any browser to be used.

In CUSS 1.2, to help provide a more consistent and predictable, the CUSS Technical Specification is mandating a specific browser technology and environment as the default environment.

CUSS 1.3 defines a Standard CUSS Browser. This standard browser is Microsoft Internet Explorer 8, which must run as the default browser environment in which browser-based applications run. The move to IE8 provides these benefits:

- Uses a browser version that is supported and updated by the vendor
- Allows CUSS sites to keep their browser up to date with security fixes
- Provides a predictable environment for all CUSS applications (compared to earlier versions of CUSS, where different sites used different browsers and versions.)
- Provide a more modern browser with better standards compliant and compatibility
- Operates correctly on modern environments like Windows 7
- Provides new features for application developers deploying into CUSS 1.3 environments

CUSS 1.3 also defined an alternate CUSS Browser, which must be deployed alongside the Standard CUSS Browser and available as an option for CUSS application providers. The alternate browser must be Google Chrome 27 or later.

This alternate browser must be the browser environment only for those applications where the application provider has specifically requested to run their application in the alternate CUSS browser when running on CUSS 1.3 kiosks.

Adding a new alternate browser to CUSS 1.3 provides these benefits:

- A more modern and standards compliant browser environment than is offered by IE8
- Continues to operate under the Windows XP environment
- Provides more capability for application suppliers wishing to deploy under CUSS 1.3
- A more responsive browser environment for heavy usage of javascript

Browser Environment (General Comments)

- The CUSS-TS does not mandate a particular build or patch of IE8. A CUSS 1.3 platform provider can deploy any updates to IE8 as needed to maintain security fixes and patches up to date.

- Platform providers *must* include the standard CUSS browser (IE8) as the default browser on the kiosk and *must also* include the alternate CUSS browser (Chrome 27+) as an available browser on the kiosk.

- A CUSS 1.3 platform provider must not accommodate any request from a CUSS application provider to install or use any version of Internet Explorer other than IE8, as this mode of operation is not support by Microsoft.

- A CUSS 1.3 platform provider must not accommodate any request from a CUSS application provider to install or used any other browser that is not Google Chrome.

- The CUSS platform provider must deploy Google Chrome Frame as a component of the alternate CUSS browser, so that application providers can opt to run with the Coogle Chrome engine inside the default CUSS browser.

    - ***Important Note:*** *At the time of publication of CUSS 1.3, Google had recently announced withdrawal of support for Google Chrome Frame as of January 2014. Application suppliers may wish to take this into account and opt to deploy in the native Google Chrome browser instead of Google Chrome Frame.*

- The CUSS platform can deploy a default browser container that is based on the IE8 engine but is not required to host the applications in the actual Internet Explorer 8 browser. This allows platforms to deploy appropriate settings and application control features as needed for application control.

- A browser-based CUSS application that does not operate correctly in the default CUSS browser (IE8) or the alternet CUSS browser (Chrome 27) is not compliant with CUSS 1.3.

- An application can deploy its own browser if and only if that browser runs entirely within the java virtual machine environment of the standard CUSS java environment.

- CUSS platform providers may choose to deploy alternate browsers, such as Firefox or Opera, on their kiosk environment alongside IE8 and Chrome 27, for example, to control custom applications such as information signage, as long as these other browsers to not interfere with CUSS applications on the kiosk running in the standard or alternate CUSS browsers.

Presentation Tools and Libraries

The use of "-" in the technology table indicates no change from the previous version of the CUSS standard. If no earlier version is defined, that technology is not supported in that version of CUSS.

CUSS 1.3 platforms must provide at least the versions listed in the "1.3 Required Version" column, and the major version of the tool must be as indicated.

New versions of tools can only be deployed as decided by the CUSS Technical Solution Group via changes or updates to this Technical Specification document. Application suppliers and providers cannot assume that a newer version of the tool will be available on kiosks even when and if an update is available from the Standards body or Organization.

An exception is allowed if an update to a new major version is needed to correct vulnerabilities, exploits, or other serious technical problems not corrected by the vendor in the major versions of the tools specified below.

As an example, given here for Adobe Flash but applicable to all CUSS technologies:

- All CUSS 1.3 kiosk platform providers must include at least Adobe Flash 11.7.7. The platform provider may choose to deploy a later point version of Adobe Flash 11
    - For example, Adobe Flash 11.7.7+, 11.8, 11.9, and other point releases of Adobe Flash 11 are permitted in a CUSS 1.3 environment and are compliant.
- The platform provider must not deploy any major release version of Adobe Flash greater than 11:
    - For example, Adobe Flash 12.0 is not permitted in a CUSS environment, and a platform that deploys Adobe Flash 12 is not a CUSS compliant site.
- An application must not be written using Flash technology that depends on a version of Flash greater than 11.7.7.
    - The application provider must be aware that it *may* be running on a point release newer than Adobe Flash 11.7.7.
    - However, an application that *requires* a version Adobe Flash greater than 11.7.7 is not a CUSS compliant application
    - An application provider cannot demand that a platform provider install a point release of Adobe Flash 11 newer than 11.7.7
    - An application provider cannot demand that a platform provider install a major release of Adobe Flash newer than 11 (for example, Flash 12 or 12.1.)

- As is the case for any business relationship, application and platform providers may discuss one-to-one options for deploying newer point releases of Adobe Flash 11 at specific sites, to accommodate application needs and also to assign ownership of the operational risk this imposes on other applications at a site.

This strict requirement is absolutely required to ensure a common, consistent understanding of what specific environment is available for CUSS applications.

Because of this, special care must be taken:

- Kiosk providers or suppliers must make sure their CUSS kiosk environment meets the CUSS 1.3 versions requirements.
- Application providers and suppliers must make sure their development teams design for and use only the versions listed, not more recent versions of the tool

The CUSS standard takes the view that having an environment and tool versions that are consistent and predicatable for *all* application providers is the priority.

Allowing individual application providers to request newer versions of tools would require imposing this change on all other application providers deployed at that site, which is not acceptable as CUSS would no longer be a predictable or consistent environment.

Kiosk PC System Requirements

Minimum Requirements:

The minimum PC requirements for a CUSS kiosk to support all the mandatory CUSS 1.3 Technologies listed above, are:

- Pentium 4 2.33 GHz or higher processor (or equivalent)
- 512MB of RAM (base platform, operating system, and a single CUSS application)
- Windows XP SP3 32-bit, or Windows 7 32-bit
- At least 1GB of available disk space per application
- Screen resolution of 1024x768 or above (1024 or more wide and 768 or more high)

These are the absolute minimum requirements that an existing kiosk enclosure must provide, in order to comply with CUSS 1.3 and provide all the tools reqired for compliance.

The platform should also provide an extra 128MB of RAM for each additional CUSS application running on the kiosk. For example, the total memory required for a kiosk running five CUSS applications is 1024 MB of RAM.

Note that *individual* applications may use up to 192MB of memory combined for all their processes. However, kiosks can be sized assuming the *average* application will not exceed 128MB.

Also, note that while these are the minimum system requirements for these required technologies, they are not necessarily the recommended system configurations for those technologies. In particular, high-definition video playback or sophisticated animations may impose significantly higher hardware requirements, or even requirements for special graphics processing hardware not required by CUSS.

Hence the bare minum is in place to give guidance as to which existing CUSS kiosks can be upgraded to CUSS 1.3. Also note that, depending on the existing work load of existing CUSS 1.0-1.2 kiosks, older kiosk hardware that does not meet these bare minimum hardware requirements may still be able to operate normally. For example, unless applications require advanced AIR or Flash/Shockwave graphics, kiosks deployed to the CUSS 1.2 requirements will continue to prove adequate for CUSS 1.3.

Recommended Requirements:

For new kiosks or kiosks being updated, CUSS-TS 1.3 recommends hardware that is above the bare minimum system requirements:

CUSS 1.3 *recommends* the following PC specifications for new deployed kiosks (as specified in the CUPPS Technical Specification revision 1.03):

- System CPU with a Passmark score of at least 2000.
- 4GB of RAM total (base platform, operating system, and all applications)
- Windows XP SP3 32-bit, or Windows 7 32-bit

Application Design for CUSS 1.3:

Many older kiosks have been deployed for four or more years and and have not been upgraded. Generally speaking, applications written to run on CUSS kiosks should not presume they will be running on fast, recent PCs, even if the kiosks are running CUSS 1.3.

In addition, even newer kiosks may be running systems that are not as fast as typical current desktop PC standards. This can be for a variety of reasons, many related to industrial design and maintenance needs of PCs included inside kiosks.

As such, airline application providers should review the correct operation and performance of their application on the range of PC equipment on which they expect the application to run, and evaluate if any changes are needed. For example, applications should not necessarily rely on very fast systems for any reason, such as for advanced visual effects or animations.

In addition, airline applications should not presume any screen resolution larger than 1024 pixels wide and 768 pixels high. While most newer kiosks support larger resolutions, many existing kiosks only support 1024x768, and applications would need to accommodate running in that resolution.

Presentation Service Technology	Standards body or Organization	1.1 Required Version	1.2 Required Version	1.3 Required Version *(Major version required, later minor version allowed)*
Adobe Air	Adobe	-	1.5	3.7
AEA (Printers)	AEA[1]	AEA99	AEA2008	AEA2009
AEA (Self Bag Drop)	AEA	-	-	AEA2012-2
Flash Movies	Adobe Flash[2]	7.0	10.0[3]	11.7.7
Standard CUSS Java Browser Plug-in	Oracle	Java 1.3.1_06 (default) Java 1.5.0_04 (available)	Java 1.3.1_06 (default) Java 1.3.1_20 (available) Java 1.5.0_04 (available)	Java 7u21
Standard CUSS Java Command Line	Oracle	Java 1.3.1_06 (default) Java 1.5.0_04 (available)	Java 1.3.1_06 (default) Java 1.3.1_20 (available) Java 1.5.0_04 (available) JRE6u12 (available)	Java 7u21
JavaScript or ECMA-262	Netscape/Mozilla, ECMA	-	1.5	JScript 5.8 (IE8) 1.8.5 (Chrome)
Macromedia Audio	Adobe Shockwave	10.1	11.0	12.0
Macromedia Movies	Adobe Shockwave	10.1	11.0	12.0
PDF	Adobe	-	1.7 (ISO 32000)	1.7 (ISO 32000)
Quicktime	Apple	6.5	7.5.5	7.7.3
SilverLight	Microsoft	-	2.0	5.1
SVG (printing)	W3C	1.1	-	-
Standard CUSS Web Browser	Microsoft	-	-	IE8
Alternate Web Browser[4]	Google	-	-	Chrome 27 Chrome Frame 21

[1] If an AEA printer cannot be upgraded to the correct version of AEA, then any kiosk that uses that printer cannot compliant with that version of CUSS.

[2] Please review http://www.adobe.com/devnet/flashplayer/articles/fplayer10_security_changes.html for compatibility information.

[3] Applications that used Native Interface (DLL) portions of Flash 6 or Flash 7 are not supported in CUSS 1.2+ as they are not compliant with CUSS

[4] The CUSS platform must provide both the standard and the alternate browser environments. CUSS applications can choose which of the two to use.

What other Software can an application use?

CUSS is a standard designed to run on multiple operating systems, and to remain interoperable across a well-defined but fixed set of technologies. For this reason, CUSS applications running on a CUSS kiosk must be java-based or browser-based and can only use the common tools listed above, to ensure that they function on all platforms.

This approach is in place to ensure compatibility across platforms, providers, and kiosk sites, to ensure the integrity of kiosk provider's kiosk image and configuration, and to allow the kiosk provider to fully control the kiosk image in accordance with the CUSS Technologies list.

As such, CUSS application providers **cannot** demand that special operating system tools, libraries, or frameworks be installed on a kiosk to allow their application to be deployed. Restricted are such items as native .EXE programs, .DLL libraries, OS frameworks such as .NET, or any other non-application tool that would need to be installed into or copied to the kiosk image.

Any additional components required by the application, must be approved by the CUSS Technical Group and be listed in the CUSS Technologies List, possible in a new revision of the CUSS standard. The platform providers must be able to install and independently test the components to ensure they do not interfere with other parts of the platform. For example, in CUSS 1.2, the Microsoft SilverLight technology was added to the list.

For example, these are activities that are not permitted:

- Installing DLLs or EXEs in the application directory (such as SWT, a toolkit which requires JNI libraries); installing DLLs or EXEs in system or component directories not owned by the application provider (shared/airport kiosks)
- Installing custom browser plug-ins
- Including and running a customer Java Runtime in the application directory
- Installing a specific native toolkit or API (such as .NET framework)

These activities are permitted:

- Installing a font resource (.TTF) so that it is available in java or the browser
- Requesting multi-language support "packages" be available on the kiosk, for extended language support
- Include and use java libraries and toolkits as part of the application (provided or course they do not require JNI components) such as xerces, log4j, etc.
- Installation of security certificates required for secure communication

It is the responsibility of the application provider to properly abide by the licensing, terms and conditions of any font, libraries, toolkits or other resources provided for or included with an application when running on a kiosk.

Kiosk or Site-Specific Configuration for Applications

Because the CUSS standard is designed to allow airline applications to work consistently across all kiosks and platform vendors, there should be minimal need for an airline to have kiosk, station, or vendor-specific logic and configuration within application.

When and if site-specific setup or configuration or an application is required, the integration and ongoing maintenance of the application is more difficult: it requires more documentation, coordination, and tracking than other applications, and can make initial setup more difficult. Overall, it can also affect the perception that CUSS is vendor-independent and interoperable.

For these reasons, the recommended practice for application suppliers and providers is to minimize and, if possible, eliminate any site-specific settings or configurations required for their application *at the kiosk*. (At the application server or host, of course, many site-specific rules and settings are used.)

1. Applications should not use local configuration to set and find which devices to use on a particular kiosk. This should be done in code, using the guidelines from Chapters 7, 4 and 5.

2. Keep local configuration to the bare minimum, if used at all. Such configuration should only be used for known kiosk or device interoperability problems.

3. Business logic should not be set from local configuration. Instead, the application or server logic should use the kiosk name and station code from the CUSS environment to look up rules or configuration from a central configuration maintained by the airline.

4. Any local configuration that is needed due to a problem with vendor interoperability or with the CUSS Technical Standard should be brought to the attention of the CUSS Solution Working Group for discussion and revolution.

Application Technologies at the server

There are no restrictions whatsoever on any aspects of a CUSS application architecture that do not run on the CUSS kiosk itself. Application providers are free to implement any approach they

choose and any tools and technologies they need so long as the application components running **on the kiosk** abide by the restrictions outlined here.

Technologies used by the platform

The goal of CUSS is to maintain interoperability for CUSS applications across different CUSS vendors and kiosks. CUSS platforms do not themselves have to interoperate – other than of course offering a complete and correct implemention the CUSS standard itself.

For this reason, CUSS platforms and kiosks can use any tools, operating systems, frameworks, or architecture needed to implement the CUSS standard and run their platform services, tools, and other offerings without restriction, so long as nothing in the platform environment conflicts with the standard, interoperable environment required for CUSS applications (as listed above.)

Resource Limits per Application

To ensure that individual CUSS applications do not impact the resources on a shared common use kiosk, here are the resource limits that CUSS 1.3 imposes on CUSS application. These limits are increased from previous versions, taking into account the increased requirements of some newer tools and technologies.

- The application must not use more than **1024MB in its local Storage directory** on the kiosk (increased from 100MB in CUSS 1.2).

- The application process and all child processes **cannot use more than 192MB of RAM**, combined (increased from 128MB in CUSS 1.2)

- To allow the application to use up to 192MB in a java process, the platform must set the java heap size limit for applications to 256MB, either in the command line parameter configuration for launching the application, or in the global web browser java plug-in settings for all applications.

- Applications are **permitted to create additional processes**, but must clean up those processes when complete.

Applications must **not excessively use the CPU** (or “peg”) the CPU during transactions or when idle. A platform provider can deem the CPU usage “excessive” if under its reasonable evaluation, the CPU usage of the application affects the ability of other services, tools, or applications on the kiosk to run properly and responsively.

Appx F: Self-Certification Criteria

Refer to the separate document entitled *Self-Certification Criteria*, available among the CUSS certification documents from the IATA CUSS Manual. The current version is 1.1 (February 2004.)

There are no changes to Self-Certification Criteria for CUSS Technical Specification 1.3. The IATA Passenger Experience Management Group intends to publish additional testing, compliance and self-certification guidelines for CUSS applications in the 18 months following publication of CUSS-TS 1.3.

Appx G: Printer Stock and Document Types

To ensure the ability to reliably detect and use the document Feeders available on a CUSS kiosk, this Appendix collects the clarifications and specifications relating to the physical appearance and characteristics of paper documents available on the printer.

The CUSS standard defines three types of document: BoardingPass, Ticket, and GeneralPurpose, as well as BaggageTag. The characteristics and meanings of the various documents media types defined in CUSS are (from Addendum A.1.51):

1. Boarding Passes are blank paper, may or may not have magnetic stripe, may or may not have a water mark.
2. Tickets are a controlled document that have stock control numbers (SCN) and are pre-printed ticket stock as per IATA RP. It is unlikely that a CUSS kiosk operating in a shared environment would contain ticket stock as the SCN is typically per airline and pre-printed.
3. General Purpose Documents are blank paper.
4. Baggage Tags are documents that meet the CUSS bag tag specification
5. The paper may or may not be perforated.
6. The other types are optional. Stock use on a dedicated (non-shared) CUSS kiosks may be customized as needed.
7. AEA and GPP printers must support all the requirements of IATA Resolution 792 for bar coded boarding passes, including a minimum print resolution of 200dpi.

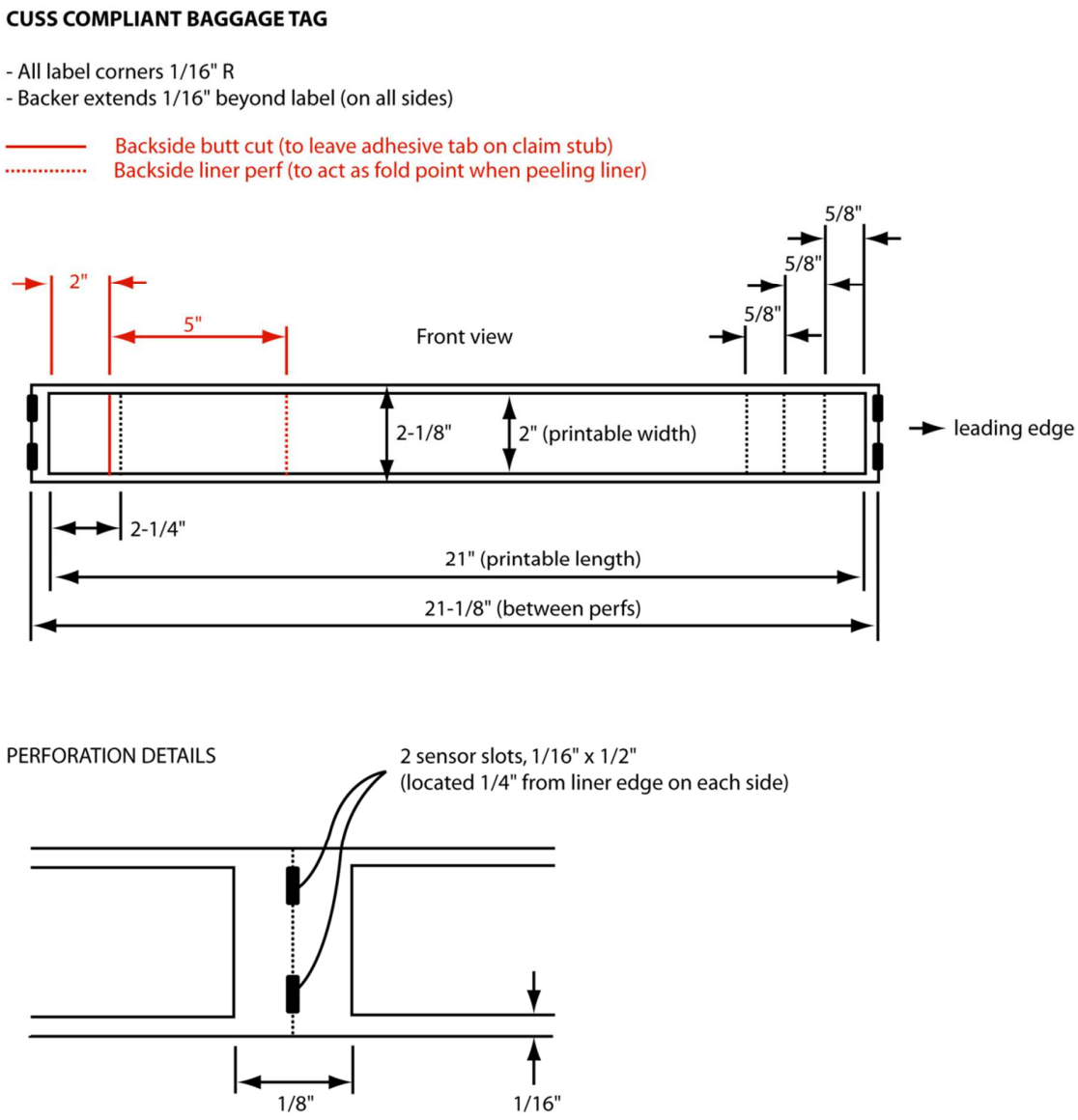
Of the three, only BoardingPass is mandatory and all other types are optional. As such, to be portable, CUSS applications must be written assuming that at a minimum, a single Feeder with BoardingPass stock is available that supports the AEA printer language and, likewise, a CUSS platform must always offer at least one component that supports AEA printing on BoardingPass stock.

CUSS 1.2 does not define the content appearance of the reverse (back) of the paper stock and does not define any way within the standard for double-sided printing.

CUSS 21” standard bag tag schematics

This section is based on CUSS 1.0 Addendum A.1.13.

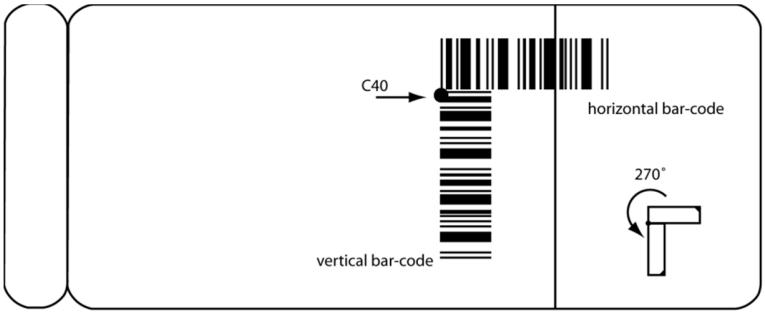
Here is a graphical representation of the standard CUSS 21” command baggage tag stock. There are no guidelines regarding what text/forms appear on the back label of the stock, if any.



BoardingPass and Ticket ATB stock layout and perforation

This section is based on CUSS 1.0 Addendum A.1.1.

BoardingPass and Ticket documents must be in 8” ATB format with binding stub. The binding stub is a non-printable area according to the AEA standards. The document may or may not be an ATB2 magnetic card, and can be from a wide range of sources (fanfold cardstock, roll thermal paper, etc.)



8" ATB document (with binding stub)

The CUSS recommended practice is to provide a 2” perforation for ATB stock, and a 3.5” perforation for GPP paper. However, both devices might be implemented on the same physical printer and paper source.

To allow airline applications to determine where the physical stub perforation actually is, the modelNumber Characteristic of the Manufacturer component of the Feeder devices can contain one of the following strings that define that paper’s perforation location. The default value is 2, and the only valid values are 2 and 3.5.

- Perf=2 (for 2” ATB stubs)
- Perf=3.5 (for 3.5” GPP stubs)

Different types of BoardingPass and Ticket stock

This section is based on CUSS 1.0 Addendum A.1.3 and A.1.46.

The CUSS standard does not indicate more than one type of boarding pass stock. Some airports may wish to offer separate stock for status customers, which could be used by applications to print out appropriate boarding passes for their tiered customers.

To allow this extra specification, if required, the platform can include one or more of the following keywords in the modelNumber Characteristic of the Manufacturer characteristics of the Feeder component of a boarding pass printer. If none are specified, then the application must assume that only normal or generic blank stock is provided.

- ECONOMY
- BUSINESS
- FIRST
- WALLET

This standard also makes the statement that there should be one virtual component per real component per function per media type. This statement means that a single virtual media and feeder component should be present for all physical bins containing identical stock. The CUSS platform must manage the physical bins and their status in a fashion that allows this single virtual component implementation.

This requirement simplifies the implementation for CUSS applications: an application only needs to find and use single component that meets its printing needs, instead of anticipating and managing multiple components offering the same thing.

This platform requirement does not apply any time the stock in multiple bins is not truly identical, such as boarding pass stock with magnetic strip or without, with wallet or without, etc. In this case, separate virtual components will exist and must have the correct characteristics set to indicate the features of the stock.

For example, if a two-feeder ATB2 printer has identical stock on both bins, then there should only be one CUSS Feeder component for that printer. If, on the other hand, one bin contains normal boarding stock and the other contains First Class stock with ticket wallet, there should be two CUSS Feeder components for that printer (the second one with the “FIRST WALLET” indicator) since the stock are not identical.

Support for Numbered (controlled) documents

This section is based on CUSS 1.0 Addendum A.1.49.

Some kiosk devices may support controlled documents (uniquely numbered, etc), but the CUSS standard does not directly allow this control information to be read by an application.

If and when the CUSS platform is able to detect and provide this information, it shall populate the serialNumber Characteristic of the Feeder component that provides the controlled documents with a document control number, with the following indication:

CTRL:<document_id>

The controlled document ID is the string following "CTRL:" up to the first space or the end of the string. If the feeder does not have controlled documents, this tag is not included. If the feeder has controlled documents but is empty, it must include the string "CTRL:-1".

If the physical device is able to query the document number directly (OCR, scanning, etc) then the actual value of the current document shall be stored in the characteristic. Otherwise, the platform must ensure that the serial number of the current document to be printed is correct and incremented automatically after successful printing (through manual reconciliation when refilling stock, incrementing counters, etc.)

Transfer Type for legacy ATB2 printers

This section is based on CUSS 1.0 Addendum A.1.7.

The CUSS 1.0 standard omits the characteristic required to determine if a printer is direct thermal or thermal transfer (the media transfer type is specified as an enumeration, but does not have a corresponding attribute in the IDL.)

To overcome this erratum in the IDL, the CUSS platform shall include the string "DirectThermal" or "ThermalTransfer" within the modelNumber Manufacturer Characteristic of the ATB2 printer components. This allows applications that need to do ATB2 revalidation or other ticket functions to detect the transfer type as required.

2-sided Document Printing

Kiosks may include printer hardware that supports printing on both sides of the page. This capability can be made available to the application, or can be restricted to built-in platform operations (such as printing advertising or other material on the back of each boarding pass.) For

more information on how this is done, please review Section 6.4.3: Reverse/2-sided printing on GPPs.

Self Bag Drop (SBD) Heavy Tag Printing

The deployment of Self Bag Drop positions may include the requirement to print or encode documents called “Heavy Tags”. These heavy tags may include a barcode, RFID tag, or other information printed on a tag or adhesive document.

IATA Resolution 740 Attachment “O” discusses heavy tag printing but does not define a printing standard for machine generated heavy tags. At the time of publication of the CUSS 1.3 specification, there is no other known standard for heavy tag documents elsewhere in the industry (IATA, ICAO, ISO, or similar.)

For this reason, the CUSS Technical Specification cannot document or support any particular document media or layout for heavy tag printing.

The CUSS Technical Specification only provides for a separate GPP printer definition that can support arbitrary printing, which may be used for Heavy Tag printing, and cannot prescribe exact document layout and formatting

Hence there is an additional burden on CUSS SBD application and platform providers to support heavy tag printing that from site to site may have substantially different requirements. This undermines a key goal of Common Use and it is unfortunately not something that this Technical Specification can yet resolve.

It is recommended that implementations of SBD keep IATA PEMG informed of various local requirements and solutions for Heavy Tag printing,

CUSS platforms at sites that need to provide Heavy Tag Printing should:

- Implement a CUSS GPP component for the heavy tag printer
- Indicate the DS_TYPES_HEAVYTAG capability in that printer’s Characteristics
- Set the proper document width and height values in the printer’s Characteristics
- Make available to airlines any layout and information requirements for heavy tags

CUSS applications that need to perform Heavy Tag Printing should:

- Look for a CUSS GPP component for heavy tag printing, by seeking a GPP component indicating the DS_TYPES_HEAVYTAG Characteristic
- Review the width and height information for this GPP’s documents
- Review the layout and information requirements made available by the airport
- Perform application business logic to determine when and how to use the CUSS GPP heavy tag printer as part of a bag drop transaction

- Consider whether reprinting tags using the bag tag printer on the kiosks to include heavy/weight information is a viable alternative to printing dedicated heavy tags.

No samples of heavy tags are available at time of publication. As they are not yet covered by an industry standard, application providers and platform providers should clearly communicate any assumptions, expections, and regulatory requirements surrounding heavy tag or other specialty document printing.

Appx H: Extended Data Type List (DS_TYPES)

Many of these values are also listed in codes.idl in the datastatus class.

Identifier (DS_TYPES)	Description	Data Format	Setup Parameters
DS_TYPES_ISO – 0	Default encoding (passport/card track data)	Refer to correct sections in this document.	None
DS_TYPES_FOID_ISO – 100	ISO track data with FOID Data truncation	Refer to Chapter 8 for more information	
DS_TYPES_PAYMENT_ISO – 200	ISO track data without truncation	Refer to Chapter 8 for more information	Comma-list of of IINs to accept
DS_TYPES_DISCRETIONARY_ISO – 300	ISO track data with DISCRETIONARY Data truncation	Refer to Chapter 8 for more information	Comma-list of of IINs to accept
DS_TYPES_FOID_JIS2 – 14100	JIS-2 track data with FOID Data truncation	Refer to Chapter 8 for more information	
DS_TYPES_PAYMENT_JIS2 – 14200	JIS-2 track data without truncation	Refer to Chapter 8 for more information	Comma-list of of IINs to accept
DS_TYPES_DISCRETIONARY_JIS2 – 14300	JIS-2 track data with DISCRETIONARY Data truncation	Refer to Chapter 8 for more information	Comma-list of of IINs to accept
DS_TYPES_VING – 1000	VING magnetic card encoded keylock data format	See VING vendor specification document	None
DS_TYPES_TESSA – 2000	TESSA magnetic card encoded keylock data format	See TESSA vendor specification document	None
DS_TYPES_SAFLOK – 3000	SAFLOK magnetic card encoded keylock data format	See SAFLOK vendor specification document	None
DS_TYPES_TIMELOX – 4000	TIMELOX magnetic card encoded keylock data format	See TIMELOX vendor specification document	None
DS_TYPES_KABA_ILCO – 5000	KABA iLco magnetic card encoded keylock data format	See KABA vendor specification document	None
DS_TYPES_KABA_ILCO_FOLIO – 6000	KABA FOLIO magnetic card encoded keylock data format	See KABA vendor specification document	None
DS_TYPES_IMAGE_IR – 7000	Infrared biometric or document image	Industry standard JPG or BMP	"JPG" (default) or "BMP"
DS_TYPES_IMAGE_VIS – 8000	Visible biometric or document image	Industry standard JPG or BMP	"JPG" (default) or "BMP"
DS_TYPES_IMAGE_UV – 9000	Ultraviolet biometric or document image	Industry standard JPG or BMP	"JPG" (default) or "BMP"
DS_TYPES_IMAGE_PHOTO – 10000	Photographic biometric or document image	Industry standard JPG or BMP	"JPG" (default) or "BMP"
DS_TYPES_IMAGE_COAX – 11000	Coaxial biometric or document image	Industry standard JPG or BMP	"JPG" (default) or "BMP"
DS_TYPES_CODELINE – 12000	OCR data detected within document and decoded	ASCII text	None
DS_TYPES_BARCODE – 13000	Barcode data detected within document and decoded	ASCII text or binary stream	None
DS_TYPES_MIWA – 14000[50]	MIWA magnetic card ended keylock data format	See MIWA specification document	None
DS_TYPES_JIS2 – 14000	Japanese Industrial Standard card encoding type JIS2	ASCII text, single track	None
DS_TYPES_SCAN_PDF417 – 15000	Scanner/CCD support for PDF417 encoded data	Binary data, multiple track	None
DS_TYPES_SCAN_AZTEC – 15100	Scanner/CCD support for Aztec encoded data	Binary data, multiple track	None

[50] This conflicts with DS_TYPES_JIS2 for backwards compatibility reasons.

DS_TYPES_SCAN_DMATRIX – 15200	Scanner/CCD support for DataMatric encoded data	Binary data, multiple track	None
DS_TYPES_SCAN_QR – 15300	Scanner/CCD support for QR encoded data	Binary data, multiple track	None
DS_TYPES_SCAN_CODE39 – 15400	Scanner/CCD support for Code 3 of 9 1D barcodes	Text data, multiple track	None
DS_TYPES_SCAN_CODE128 – 15500	Scanner/CCD support for Code 128 1D barcodes	Text data, multiple track	None
DS_TYPES_SCAN_CODE2OF5 – 15600	Scanner/CCD support for Interleaved 2 of 5 1D barcodes	Text data, multiple track	None
DS_TYPES_ISO7816 – 16000	Communication protocol for PICC/NFC/RFID device	See Section 7.15	See Section 7.15
DS_TYPES_PRINT_2S_PAGE – 16100	GPP support for back-side printing a single page	See Section 6.4.3	See Section 6.4.3
DS_TYPES_PRINT_2S_MULTI – 16200	GPP support for for front-back printing of multi-page documents	See Section 6.4.3	See Section 6.4.3
DS_TYPES_PRINT_PDF – 16300	PDF version 7.0 / ISO32000 compatible print data	See PDF specification and Section 6.4.2	See Section 6.4.2
DS_TYPES_MIFARE – 17000	Communication protocol for PICC/NFC/RFID device	See Section 7.15	See Section 7.15
DS_TYPES_SUICA – 17010	Communication protocol for PICC/NFC/RFID device	See Section 7.15	See Section 7.15
DS_TYPES_ISO15961 – 18000	IATA RFID baggage tag device	Refer to CUSS.SBD.XSD	
DS_TYPES_RP1745 – 18010	IATA Baggage Service message format	Refer to CUSS.SBD.XSD	
DS_TYPES_WEIGHT – 18020	Baggage weight/load from Scale or SBD device	Refer to CUSS.SBD.XSD	
DS_TYPES_HEAVYTAG – 18030	Indicates a printer designated as a baggage heavy tag printer	None	
DS_TYPES_SBDAEA – 18040	AEA-SBD control language	Refer to AEA2012-2	
DS_TYPES_SBDCUSS – 18050	CUSS-SBD control language		
DS_TYPES_EPASSPORT_DG1 – 20100	ICAO e-Passport RFID data DG1	Binary data – see ICAO specification	None
DS_TYPES_EPASSPORT_DG2 – 20200	ICAO e-Passport RFID data DG2 […]	Binary data – see ICAO specification	None
...DG3-DG20 – 20300-2200	ICAO e-Passport RFID data DG20	Binary data – see ICAO specification	None
DS_TYPES_EPAYMENT – 2300	Generic Payment and EMV Transaction CUSS 1.3 format	Refer to CUSS.PAYMENT.XSD	

Appx I: Application Updates and Distribution

One of the main issues surrounding the deployments of kiosks and applications using the CUSS 1.0 and 1.1 standards is that the requirements to packaging and distributing applications is not consistent, and different groups have a different understanding as to what types of changes require re-testing, and to what extent.

While IATA has not yet defined or recommended any application or platform change management procedure for vendors and providers to follow, CUSS 1.2 now includes some overall guidelines that will make the existing situation smoother.

Packaging and Distribution of CUSS Applications

This section is taken from CUSS 1.0 Addendum A.1.22.

This section provides information on how a CUSS application provider should bundle and deliver an application to a kiosk provider (for a new application or an application update.) It provides a simple, common packaging method that is not tied to any application provider or platform/kiosk vendor tools. An application provider can then create and send a common, consistent update to all kiosk sites.

Kiosk providers or sites are free to repackage any application updates they receive in any fashion suitable to their internal tools, processes, and operating environment. The application provider should not be involved in this process, however, as it is (usually) proprietary to the kiosk vendor. More specifically:

An airline is not required to create, provide, or update any application update information that is specific and proprietary to the airport kiosk being updated. It only needs to provide the basic information listed below. The kiosk provider must integrate these airline files into whatever file, configuration, and update management/distribution methods their kiosks or platform provider use, without input from the airlines.

The goal is to ensure that an airline can easily deliver a standard update to numerous airports at the same time, and it is up to the airport, not the airline, to process that update and integrate it with their kiosks using whatever proprietary processes they have chosen to use.

An airline will only provide the following items when delivering a CUSS application. No where should there be any reference to specific kiosk or vendor files or directory locations.

1. ZIP archive (including subdirectories, if needed) of the entire application and all its support files which must be installed on the kiosk in order for the application to run. This archive includes only files (software, resources, configuration) for the airline application,

and cannot contain any platform files, platform directories, platform tools, or platform configuration files.

2. *If the application is browser-based:* The startup URL must be provided (either over the network, or a local HTML installed by the application archive.) If any specific values are needed in the URL, the startup URL must include these parameters. For example, this could include the Storage Component location, the Kiosk Name, etc.

3. *If the application is java-based:* The full startup command line for the application must be provided. This must include any java runtime parameters, classpath statements, and class parameters.

4. Files or additional instructions for installation (such as new system font files, or language packs) only as permitted in Appendix A (Application Technologies.)

5. The version number of the application and whether it required a CUSS 1.0, CUSS 1.1, or CUSS 1.2 environment.

6. The company code and application name used by the airline application when communicating with the platform using the akID structure.

7. Application change/revision history with sufficient information for the airport to determine if any additional integration testing is needed (if the airport SLA requires this information.)

8. *Recommended for ease of troubleshooting:* A README or similar document that explains how to see if the application is running properly (log locations, etc) to provide assistance during deployment.

An airline is not required to provide proprietary update scripts, install processes or incremental patch files, or detailed description of all files/directories (though this may be required for initial integration.)

CUSS Certification and Re-Certification Guidelines

This section is taken from the March 2007 CUSSMG (now PEMG) presentation *CUSS Certification and Re-Certification Guidelines v2*.

Many airlines have integrated and certified their IATA CUSS application with various CUSS providers. These applications are now deployed across many CUSS sites. Similarly, many CUSS platform providers have integrated and certified their IATA CUSS platforms with various CUSS application providers. These platforms are now deployed across many CUSS sites.

What are the guidelines regarding re-certification with IATA or re-integration with the providers as the airline continues to evolve their CUSS application with new functionality and enhancements? Similarly, what are the guidelines regarding re-certification with IATA or re-integration with the airline application providers as the platform provider continues to evolve their CUSS platform with new functionality and enhancements?

Below is a table of changes and change definitions to assist airlines and airports and others interested in the CUSS process in determining when CUSS integration or full certification is required after implementing changes. Airlines, airports and providers can use these guidelines in planning application updates and distribution planning.

CUSS 1.3 defines an Automatic Remote Update (ARU) procedure. Only certain types of changes are elegible for automated updates. Please review the ARU Chapter 9 as well as these change definitions to be clear when and how changes to your application can be deployed.

Application Change Definitions

Change Level	Description
1	A level 1 change dictates that CUSS certification or re-certification of the application is required. The CUSS certification document defines the types of changes where certification/re-certification is required. **Changes of this type ARE NOT eligible for Automated Remote Updates.**
2	A level 2 change dictates that integration or platform end-to-end testing is required with the airline application provider in the lab. A change is defined as level 2 if there is a significant change in how the application interfaces with the platform. **Changes of this type ARE NOT eligible for Automated Remote Updates.**
3	A level 3 change indicates a complex deployment where site coordination and configuration management is required. A remote test of the application is required. The changes in the application impact how the application interfaces with the platform, such as a change in the sequence of calls to the platform. **Changes of this type ARE eligible for Automated Remote Updates provided they are first distributed to a limited "beta site" selection of kiosks in a production environment.**
4	A level 4 change is a low risk/minor change. The changes in the application do not affect interaction with the CUSS platform. **Changes of this type ARE eligible for Automated Remote Updates. ARU Validation must be performed at all sites.**

Application Change Examples (with corresponding level)

This list is not a comprehensive list and is not intended to circumvent proper change management, testing practices etc.

Description of Change	Level
Add support for a new media or data device to the application (e.g. new peripheral) such as adding ATB/2 reading, passport reading or bag tag printing	1
Change in Technical Specification version required by the application	1
The first implementation of a new application	1
1st time deployment of the application on a different vendor platform	2
The application provider changes the toolkit vendor used to interface with the platform	2
Change in the application architecture. Examples: A change from synchronous to asynchronous calls or from a local hosted ORB to a remote ORB.	2
Configuration changes that require site validation and operator intervention	2
Application business flow changes that reuse preexisting platform interface	3
Add/remove files and directories in the airline specific application file/directory root (shared libraries see above)	3
Changing the technologies that you use in your application – such as adding flash animation or changing the ORB	3
Change in screen resolution for the application – requires some coordination with platform to confirm support, resolution change, etc.	3
Configuration changes that do not require site validation	3
Change in startup parameters or startup URL	3
Change in network routing, port usage, firewall, host connection, host architecture, etc.	3
Change common launch configuration - new button, new brand, different airline code or company code, etc.	3
Change in pectabs, logos, templates or other printer resources as well as cosmetic changes to existing screens, images, sounds, languages residing in the local airline applications	4
Change business logic in the application server	4
Change host logic	4
Configuration changes –server based	4

Platform Change Definitions

Note that any ARU guidelines apply only to updates to CUSS applications. Changes to the platform are not subject to ARU restrictions or definitions.

Change Level	Description
1	A level 1 change dictates that CUSS certification or re-certification of the platform is required. The CUSS certification document defines the types of changes where certification/re-certification is required. Communication with the airlines is required.
2	A level 2 change dictates that integration or platform end-to-end testing is required with the affected airline applications in the lab. Communication with the airlines is required.
3	A level 3 change indicates a complex deployment where site coordination and configuration management is required. A remote test of the application may be required. Communication with the airlines is required.
4	A level 4 change is a low risk/minor change. The change does not affect interaction with the CUSS applications. Communication with the airlines is *recommended*.

Platform Change Examples (with corresponding level)

This list is not a comprehensive list and is not intended to circumvent proper change management, testing practices etc.

Description of Change	Level
New platform	1
Adding new device types to the platform (not already certified)	1
Major change in CUSS version supported (i.e. 1.0 to 2.0)	1
Activating new device types in the platform at the airport	2
Implementing IATA CUSS minor releases (ie 1.2 to 1.3); specifically ones that alter device or platform behaviour	2
Changes to the networking architecture in the airport	2
Changing the default screen resolution	2
Changing the operating system (i.e. XP to Windows 7)	2
Changing browser technologies	2
Tangible increase in platform instability or reduced reliability in applications (when no changes occurred in the application)	2
Changes to the ORB the platform uses	2
Minor change in CUSS version supported (i.e. 1.0 to 1.1) that do not alter device behaviour	3
Adding new device vendor – i.e. switching boarding pass printer supplier from supplier A to supplier B	3
Installing Microsoft patches or other operating system updates	3
Changes to the common launch application such as changing the branding or adding a 2nd page to the CLA screens	3
Change in application paths or how they are configured	3
Changing the list of supported screen resolutions	3
Adding or modifying kiosk IDs in the network; changing location codes	3
Changing paper stock such as from magnetic to non-magnetic or from thermal to non-thermal	3
Adding new kiosks in the airport	4
Adding new CUSS applications to the airport installation	4
Upgrading existing CUSS applications to the airport installation	4
Additional functionality in the platform that doesn't impact CUSS interfaces	4

Appx J: Upgrading to a new version of CUSS

The CUSS Technical Standard update from version 1.0/1.1/1.2 to 1.3 is designed to allow all existing CUSS applications, with minimal if any changes, to continue to work on CUSS 1.3 environments, while making available to new applications new features and technologies.

Platforms running CUSS 1.3 must be updated to include the new features introduced in the new version of the standard, as well as offer an updated kiosk environment that provides the new technologies available to applications in CUSS 1.3.

This Appendix provides a “quick reference” regarding the tasks facing platform and application developers who intend to upgrade to the latest version of CUSS. It does not replace the rest of this specification document: application and platform suppliers should review and understand all changes in this updated specification and determine what effect the changes have on their own platform or application software.

Updating Applications for CUSS 1.3

Some applications may need to change in order to fix fundamental compatibility problems identified and solved within the CUSS 1.0 standard between 2003 and 2011. These fixes have been documented in regular updates to the CUSS 1.0 Addendum (see below) and the goal is to unify these corrects in a single, consistent implementation across all kiosk platforms running the CUSS 1.2 specification. CUSS 1.3 adds additional changes and features on top of CUSS 1.2

Any changes to application behaviour are due to the nature of the issue being fixed. Where the CUSS standard was unclear about how an aspect of the specification functioned, in some cases different platforms and applications implemented the specification in incompatible ways. Because of this, once the problem had been resolved by choosing one or the other of those incompatible interpretations, the remaining platforms and applications need to change in order to operate properly.

Many applications will not need to change at all, if they are not affected by these previous Addendum resolutions which were part of CUSS 1.2. Existing applications can also continue to use the CUSS 1.0/1.1/1.2 technologies, as they remain available on CUSS 1.3 kiosks.

It is anticipated that most applications will not need to undergo Integration testing in order to be deployed on a CUSS 1.3-compliant kiosk environment, since any changes needed are very specific and the expected behaviour of applications and platforms is well-defined for these issues. However, the application providers may still wish to coordinate a short Integration session with various kiosk providers to verify the new behaviour.

If a CUSS application requires changes for CUSS 1.3, but will continue to run on CUSS 1.2 or earlier kiosks as well (for example, their own proprietary kiosks, for example) then the application may need some extra logic to handle the behaviour of both versions of the standard on which it will run.

For example, the sequence of events from card readers may be different on older CUSS 1.0 kiosks, and CUSS 1.2 or CUSS 1.3 kiosks, as clarified in Section 3.8 of this document. An application might need to retain its existing card reader event logic (for older platforms) but also accept the newer CUSS version.

Here is the list of changes and review required for an application to run on a CUSS 1.3 kiosk:

Review Items (CUSS 1.2 to CUSS 1.3)

1. Changes in AEA2009 related to 2D barcode encoding may require pectab or data changes. In addition, any pectabs that use barcode “4” or “V” for 1D barcodes need to be changed to “7” and “W” with the equivalent data streams.

2. Make sure the application does not check for and require specific platform or specification versions in the CUSS Environment, which would be different on a CUSS 1.3 kiosk.

3. If the CUSS application reads magnetic cards for any purpose, update the application to comply with the CUSS FOID Addendum, included in CUSS-TS 1.3 as Chapter 8.

4. Verify that your application runs in JRE7 as the sole java runtime environment. This applies to browser-based applications running the java plug-in, or traditional full java applications running from the command line.

5. If your application is browser-based, verify that it runs correctly in the Standard CUSS Browser (Microsoft Internet Explorer 8).

Review Items (CUSS 1.0/1/1 to CUSS 1.2)

1. Because AEA printers run AEA2008, any pectabs that use barcode “4” or “V” need to be changed to “7” and “W” with the equivalent data streams.

2. Review how the application uses the receive() directive and make sure it complies with Section 3.6.8.1.

3. Review how the application uses the offer() directive and monitors Dispenser standard, and make sure it complies with Sections 3.6.9.1 and 3.6.11.

4. Read Section 3.8 and determine if any changes are needed in how an application monitors and uses Media input devices such as card readers and passport scanners.

5. Make sure the application does not check for an require specific platform or specification versions in the CUSS Environment, which would be different on a CUSS 1.2 kiosk.

6. Read Section 1.7 on Data Security and check if the application requires any changes to meet the guidelines.

7. If your application uses Adobe Flash, it will be running under Adobe Flash Player 10.0. This version includes security changes that *may* impact existing Flash content written for Flash 6 or 7. For more information, please review:

http://www.adobe.com/devnet/flashplayer/articles/fplayer10_security_changes.html

New Features (CUSS 1.2 to CUSS 1.3):

1. If the application performs any sort of automated remote update feature, change it to attempt the new CUSS 1.3 ARU requirements.

2. If your application using the IntegratedConveyor interfaces from CUSS 1.1, it will need to change to use the updated conveyor interfaces defined in sections 7.16 and 7.17. Platform providers may, however, leave the existing IntegratedConveyor interface in place to ease the transition.

3. Applications that require multi-byte AEA printing may use the new EP and ES AEA commands to request extended language/codepage support from the platform. However, this support depends on the printer capabilities on the kiosk and extended languages may not be supported.

4. If your application uses a proprietary Payment Device and interface, it may need to be updated to support the new standard payment interface defined in section 7.19.

New Features (CUSS 1.0/1.1 to CUSS 1.2):

1. Review new Sections 2.4.4 and 2.4.5. There are many new application mode and notification features available to applications. Of immediate interest are Sections 2.4.5.6 and 2.4.5.7, which an application can use to provide more detailed information about what is going on. This can assist in front-line monitoring of CUSS applications.

2. Review the new technology lists and understand what is available for future versions or changes to the application (long-term upgrade planning.)

3. Explore how to use the new 2D barcodes in AEA2008 as part of the airline's barcoded boarding pass strategy (BCBP.)

Updating Platforms for CUSS 1.3

CUSS platforms must be updated to include all fixes and resolutions within the Technical Specification, and they must also implement some new features (such as Automated Remote Update events) that are now part of the standard. Updated platforms must then be rolled out into a kiosk environment that makes available to the applications the new CUSS 1.3 Technologies listed in Appendix E, such as the standard CUSS java and standard CUSS browser environments.

While it is expected that most existing applications will continue to operate without problem under CUSS 1.3, this is not guaranteed (see above.) Platform and kiosk providers should also plan and coordinate an upgrade of their existing sites to CUSS 1.3, communicating the information and timeline with their airport locations and application providers.

Required Changes:

1. Update all AEA printers to AEA2009 including support for QR, DataMatrix and Aztec 2D barcodes, EAN-13 1D barcodes, and support the ZS and AV printer commands. New commands that must also be supported are the RC and RI commands, as well as EP and ES to support requests for extended language/codepage printing.

2. All kiosk printers must support 200dpi or better printing, to comply with the requirements if IATA Resolution 792 (Bar Coded Boarding Pass.) If the printers in the kiosk to not meet this requirement the kiosk is not CUSS 1.3-compliant.

3. A CUSS 1.2 or 1.3-compliant kiosk and platform must include a barcode scanner capable of reading IATA Resolution 792 barcodes in the following symbologies: PDF417, QRCode, Aztec, Datamatrix. (Previous versions of the CUSS-TS did not require any barcode scanner – a hardware upgrade may be needed for legacy kiosks.)

4. Update the kiosk environment/image to support all the new technologies for CUSS 1.3 (such as JRE7, Adobe AIR 3.7, and similar described in Appendix E.)

5. If a kiosk printer cannot be updated to AEA2009, or the kiosk image does not include all new technologies and tools mandated by CUSS 1.3, then the kiosk is not CUSS 1.3-compliant.

6. Review Section 2.4.5 and add platform support the Automated Remote Update requests

7. Implement the CUSS FOID Addendum (Chapter 8) for all card reader interfaces.

8. Make sure the new platform version and specification versions are published to the application as described in Section 3.3.1.1.

9. Update all ACTIVE Notification logic to include the new “NOTIFICATION=” prefix for notification data.

Optional Changes:

1. If a platform is running on a kiosk with a baggage scale/conveyor, make sure it is implemented via the new Baggage System component definition listed in Section 7.16. The platform must implement both the AEA-SBD interface, as well as the CUSS Component Mode interface.

2. If the platform is running on a kiosk with an intergrated payment device (Chip & PIN, etc), make sure it is implemented view the new Payment Interface component definition listed in Section 7.19.

3. Platforms that want to add additional control and oversite of application ARU requests may choose to implement some or all of the platform features related to ARU, such as snapshot and rollback. How this is done and to what extent is a platform implementation exercise.

4. If the kiosk includes barcode scanners that support multiple reads on a single document, change the barcode interface to return additional data tracks representing multiple simultaneous reads.

Updating Platforms for CUSS 1.2

CUSS Version 1.3 is the current version, and new/updated kiosks should be updated to the latest version, and not to CUSS 1.2. However, for reference, the incremental change between CUSS 1.0/1.1 and CUSS 1.2 is provided here.

CUSS platforms must be updated to include all fixes and resolutions within the Technical Specification, and they must also implement some new features (such as Application Transfer mode) that are now part of the standard. Updated platforms must then be rolled out into a kiosk environment that makes available to the applications the new CUSS 1.2 Technologies listed in Appendix E.

While it is expected that most existing applications will continue to operate without problem under CUSS 1.2, this is not guaranteed (see above.) Platform and kiosk providers should also plan and coordinate an upgrade of their existing sites to CUSS 1.2, communicating the information and timeline with their airport locations and application providers.

Required Changes:

1. Update all AEA printers to AEA2008 including support for QR, DataMatrix and Aztec 2D barcodes, EAN-13 1D barcodes, and support the ZS and AV printer commands.

2. All kiosk printers must support 200dpi or better printing, to comply with the requirements if IATA Resolution 792 (Bar Coded Boarding Pass.) If the printers in the kiosk to not meet this requirement the kiosk is not CUSS 1.2-compliant.

3. A CUSS 1.2-compliant kiosk and platform must include a barcode scanner capable of reading IATA Resolution 792 barcodes in the following symbologies: PDF417, QRCode, Aztec, Datamatrix. (Previous versions of the CUSS-TS did not require any barcode scanner – a hardware upgrade may be needed for legacy kiosks.)

4. Update the kiosk environment/image to support all the new technologies for CUSS 1.2 (such as Adobe Air.)

5. If a kiosk printer cannot be updated to AEA2008, or the kiosk image does not include all new technologies and tools mandated by CUSS 1.2, then the kiosk is not CUSS 1.2-complient.

6. Add PDF printing support to all kiosks that have General Purpose Printers (GPPs.)

7. Review Section 2.4.4 and add platform support for the new modes of operation.

8. Review Section 2.4.5 and add platform support for special state transitions and notification strings.

9. Implement the recommended guidelines in Section 1.7 (Data Security) to help protect card data on the kiosk.

10. Update the receive() directive to purge data after the first read, and return multi-track data in the correct format (see Section 3.6.8.1.)

11. For kiosks with real printer dispensers, review Sections 3.6.9.1, 5.11 and 5.6 to make sure Dispensers and Feeders are implemented correctly (MEDIA_FULL, etc.)

12. Review Section 3.8 and verify that all MediaInput devices behave in accordance with the new event sequence guidelines.

13. Make sure the new platform version and specification versions are published to the application as described in Section 3.3.1.1.

Optional Changes:

5. Update the kiosk monitoring systems to support and indicate the new application status information described in Sections 2.4.5.6 and 2.4.5.7. This will assist in supporting and monitoring application status on the kiosks.

6. If a platform is running on a kiosk with a baggage scale/conveyor, make sure it is implemented via the new Conveyor component definition.

7. If a platform is running on a kiosk with extended data type devices, such as flatbed scanners and RFID readers, make sure they are implemented as described in Chapter 6.

CUSS 1.0/1.1 Addendum Reference Table

The CUSS 1.2 version of the specification includes many updates taken from the CUSS 1.0 Addendum document. For those already familiar with the previous version of the specification and that Addendum document, here is a table that references addendum entries with their Section in this CUSS Specification.

To be CUSS 1.2-compliant or later, a CUSS platform must implement all features referenced in the table below and updated within this CUSS 1.2 specification.

Ref	Addendum Title	CUSS 1.2
A.1.1	Perforation location on printers (GPP, ATB)	Appendix G
A.1.2	Supported version of AEA specification	Appendix D, 3.6.6
A.1.3	Multiple boarding pass stock support	Appendix G
A.1.4	Multiple-airline branding on launch screen	2.4.4
A.1.5	Single-application mode device support	2.4.5
A.1.6	Data input device identification	Chapter 6
A.1.7	Transfer type for ATB2 printers	Appendix G
A.1.8	Clarification of AEA logo support	Appendix D
A.1.9	Clarification of AEA2002 PDF417 support	Appendix D
A.1.10	Dedicated single application mode (SAM)	2.4.4, 2.4.5
A.1.11	Offer() behavior for dispenser components (AMENDED)	3.6.4, 3.6.9.1, 5.6
A.1.12	enable()/disable() behavior for media input devices	3.6.2, 3.6.3
A.1.13	CUSS bag tag stock specification schematics	Appendix G
A.1.14	Application timeout and initial screen behavior	2.4.4, 2.4.5
A.1.15	CUSS CORBA interfaces should listen on all network addresses	2.1.1
A.1.16	Language indicator when activating an application	2.4.4, 2.4.5
A.1.17	Audio virtual component support	Appendix B
A.1.18	Track data format of OCR passport data (AMENDED)	3.1.9, 3.6.8.1
A.1.19	Extended security features of passport/document readers	Chapter 6, codes.idl
A.1.20	Further clarification of AEA logo support, PCX header	Appendix D
A.1.21	AEA barcode Vertical printing position	Appendix D
A.1.22	Packaging and distribution of CUSS application updates	Appendix I
A.1.23	Minimum duration of KillTimeout parameter	2.4.4, 2.4.5, 3.3.1
A.1.24	Virtual device linking for Passport Readers	3.6.8.1, Appendix B
A.1.25	AEA barcode128 support for subtypes 128A, 128B, 128C	Appendix D
A.1.26	AEA Bagtag color printing support	3.6.6
A.1.27	Dedicated single-app mode session start indication	2.4.4, 2.4.5

A.1.28	Component realComponentName with regards to Table B.2	Appendix B
A.1.29	TCP/IP ports used by CUSS CORBA components	2.1.2
A.1.30	Behavior of akID input structure for level() directive	3.1.10
A.1.31	Use of notify() directive for STOPPED_INITIALIZE state change	2.4.1.2
A.1.32	Contents of correlation field in Event methods and fields	3.1.5, 3.1.11
A.1.33	Resolving conflicts between the IDL and the Documentation	Appendix C
A.1.34	Architecture of MediaInput devices with multiple data types	Chapter 6
A.1.35	Definition of CUSS 1.1	n/a
A.1.36	Clarification of Media behavior and Event sequence	3.7.2, 3.8
A.1.37	Behavior of components that depend on a linked component	3.2.3, 3.6.8.2
A.1.38	MEDIA_HIGH/FULL behavior of Dispenser components	3.6.11
A.1.39	Multiple documents in a single AEA data stream	Appendix D
A.1.40	Support for HTTP/1.1 protocol for browser-based applications	Appendix E
A.1.41	Behavior of multiple receive() directives after DATA_PRESENT	3.6.8.1
A.1.42	Tracking why an application is not in AVAILABLE state	2.4.4, 2.4.5
A.1.43	What barcodes does a MediaInput component support (PDF417, etc)	Chapter 6
A.1.44	Dispenser type clarification (to user, or internal feeder)	5.6
A.1.45	Amendment to A.1.34 – devices with multiple data types	Chapter 6
A.1.46	Virtual components for multiple physical bins with identical stock	Appendix G
A.1.47	Application query of the version of the CUSS platform	3.3.1.1
A.1.48	How to identify a kiosk that is an offsite (non-airport) location	3.3.1.2
A.1.49	Support for numbered (controlled) documents	Appendix G
A.1.50	Allow an ACTIVE application to transfer control to another app	2.4.4, 2.4.5
A.1.51	Clarification to BoardingPass, Ticket and General Purpose stock	Appendix G
A.1.52	What software components can an application use on a kiosk?	Appendix E

Appx K: CUSS Technical Specification Files

The entire CUSS Technical Specification includes this document and separate files. Together, all files define the current version of the CUSS technical specification. At time of publication, all individual files of the CUSS Technical Specification are available at this URL:

https://extranet2.iata.org/sites/pemg/common-use-wg/Lists/Links/AllItems.aspx

Complete File List:

IATA_CommonUseSelfService_TechnicalSpec_May2013_CUSS_1.3.pdf

IATA_CommonUseSelfService_CUSS_1.3_errata.pdf
IATA CUSS Spec_Certification_1.0-rev1.1.pdf

characteristics.idl
codes.idl
comps.idl
types.idl

CUSS.PAYMENT.XSD
CUSS.SBD.XSD

Common Use Self Service (CUSS) Technical Specification (PDF)

IATA_CommonUseSelfService_TechnicalSpec_May2013_CUSS_1.3.pdf

- Description of the behavior, environment, and requirements for CUSS compliant kiosk enclosures, platform/middleware software, and end user applications
- This document

CUSS Technical Specification Errata (PDF)

IATA_CommonUseSelfService_CUSS_1.3_errata.pdf

- List of corrections, amendments, addendums, and other changes to the CUSS Technical Specification

- Changes in this document are agreed upon by the CUSS Technical Solution Group to provide clarity to the current pblished CUSS technical specification prior to publication of the next version

- This document may also contain updates to the Tools & Technologies list for CUSS applications, should any such change be agreed by the CUSS Technical Solution Group

- This document may also contain updates to the messaging schema XSD files, should any such change be agreed by the CUSS Technical Solution Group

- The content of this errata document takes priority over any content in the other documents of the CUSS Technical Specification.

CUSS Interface Definition Language (IDL):

characteristics.idl
codes.idl
comps.idl
types.idl

- This list of requirements and verification cases that CUSS platforms and CUSS applications can perform to self-certify a basic technical compliance with the CUSS Technical Specification

- The certification testing in this document is limited to tbe base minimum requirements for operating a technically compliant application using the platform interfaces. It does not include any detail regarding application or platform architecture, design, performance, suitability, or business logic.

CUSS Payment Interface Messaging Schema (XSD):

CUSS.PAYMENT.XSD

- Defines the message format for payment transaction requests and responses between the CUSS application and the CUSS platform.

- The schema defines messages for payment requests, transactions results, itinerary information, and user transaction progress.

- XML messages encoded in accordance with this schema are distributed by CUSS internface requests such as send(), receive(), and asynchronous event notification.

CUSS Self Bag Drop Messaging Schema (XSD):

CUSS.SBD.XSD

- Defines the message format for self bag drop transaction requests and responses between the CUSS application and the CUSS platform, when the application has chose to use the CUSS-SBD component interfaces.

- The schema includes information about weight, bag dimensions, RFID tags, and other baggage-related information.

- XML messages encoded in accordance with this schema are distributed by CUSS internface requests such as send(), receive(), and asynchronous event notification.

CUSS Technical Specifications: Certification Criteria

IATA CUSS Spec_Certification_1.0-rev1.1.pdf

- The technical interface definitions of the CUSS Technical Specification, in CORBA Interface Definition Language (IDL) files.

- These definition files are used by platform and application suppliers to create interface layers using CORBA technologies, in their platforms and applications.

Glossary of Terms

AEA	Association of European Airlines
AL Application	CUSS Kiosk Airline Application
AL SM	Airline Application Provider System Manager (as defined in CUSS)
AL	Airline
AMI	Application Manager Interface (as defined in CUSS)
ANSI	American National Standards Institute
ASCII	American Standard Code for Information Interchange
ATA	American Transport Association
ATB	Automatic Ticketing and Boarding
ATM	Automated Teller Machine
AVI	Audio Video Interleaved
BMP	Bitmap (file name extension)
BP	Boarding Pass
BTP	Bag Tag Printer
CAM	CUSS Application Manager
CDROM	Compact Disk Read Only Memory
CLA	Common Launch Application (as defined in CUSS)
CORBA	Common Object Request Broker Architecture (by OMG)
CUSS	Common User Self Service standard (by IATA)
CUSSMG	CUSS Management Group (replaced by PEMG)**PEMG** Passenger Experience Management Group (replaces CUSSMG)
DCI	Device Component Interface (as defined in CUSS)
DVD	Digital Video Disc
ELI	Event Listener Interface (as defined in CUSS)
EMV	Europay, Mastercard and Visa
GPP	General Purpose Printer
GPS	Global Positioning System
HTML	Hyper Text Markup Language (by W3C)
IATA	International Air Transport Association
IDL	Interface Definition Language
IEC	International Electrotechnical Commission.
IIOP	Internet Inter-ORB Protocol
IJG	Independent JPEG Group
IOR	Interoperable Object Reference
IP	Internet Protocol
ISO	International Organization for Standardization
JFIF	JPEG File Interchange Format (file name extension)
JIS	Japanese Industrial Standard
JPEG	Joint Photographic Experts Group (also file name extension)
JVM	Java Virtual Machine
LED	Light -Emitting Diode
MIF	Management Interface (as defined in CUSS)

MPEG	Moving Picture Experts Group (also file name extension)
NC	Network Computer
OCR	Optical Character Reader
OMG	Object Management Group
ORB	Object Request Broker
PC	Personal Computer
PDF	Portable Document Format (By Adobe)
PECTAB	ParamEtriC TABle (by AEA)
PIN	Personal Identification Number
PNG	Portable Network Graphics (graphic file standard/extension)
PP	Platform Provider
RF	Radio Frequency
RFC	Request For Comment
RP	Recommended Practise (IATA)
SLA	Service Level Agreement
SM	System Manager
SMI	System Manager Interface (as defined in CUSS)
SP	Service Provider
SVG	Scalable Vector Graphics
TAT	Transitional Automated Ticket
TCP	Transmission Control Protocol
TVM	Ticket Vending Machine
UPS	Uninterruptible Power Supply
UTF	Unicode Transformation Format
VPN	Virtual Private Network
W3C	World Wide Web Consortium
XHTML	Extensible Hypertext Markup Language
XML	eXtensible Markup Language (By W3C)