

NextGenATMIA
IMPLEMENTATION GUIDE
2020



For Next Gen API App ATMs



11/2/2020

Version 2.0 Public Draft, M. Ficken

The ATMIA anti-trust and IPR policies will apply to this document (see references 5 and 6).

1 Table of Contents

1	Table of Contents.....	2
2	Introduction	9
3	Approach.....	10
3.1	Implementation Committee	11
3.2	FUNCTIONAL Working Group.....	12
3.3	TECHNICAL Working Group.....	12
3.4	CERTIFICATION Working Group	12
3.5	Proof of Concept (PoC)	13
3.6	Next-Gen ATM Journey.....	13
4	NextGenATMIA Blueprint (version 2.0)	14
4.1	Bigger Picture	15
4.2	Blueprint – ATM end-point configurations	16
5	FUNCTIONAL Specifications	17
	FUNCTIONAL Overview	18
	NON-FUNCTIONAL Overview	18
5.1	CASH-OUT Functionality (version 1.0)	19
5.1.1	COD functionality	21
5.1.2	ATM Device functionality.....	21
5.1.3	APP Service functionality	21
5.1.4	INFRAstructure functionality	21
5.2	CASH-IN Deposit Functionality (version 2.0).....	22
5.2.1	COD functionality	24
5.2.2	ATM Device functionality.....	24
5.2.3	APP Service functionality	24
5.2.4	INFRAstructure functionality	24
5.3	CRYPTO CURRENCY – BUY Functionality (version 2.0).....	25
5.3.1	COD functionality	26
5.3.2	ATM Device functionality.....	26
5.3.3	APP Service functionality	26
5.3.4	INFRAstructure functionality	26

5.4	CRYPTO CURRENCY – SELL Functionality (version 2.0)	27
5.4.1	COD functionality	28
5.4.2	ATM Device functionality	28
5.4.3	APP Service functionality	28
5.4.4	INFRAstructure functionality	28
6	TECHNICAL Specifications	29
6.1	TECHNICAL CASH-OUT PROCESS FLOW	30
6.1.1	Interface-1 (COD – ATM Appliance)	31
6.1.2	Interface-2 (ATM Appliance – NextGenATMIA APP)	31
6.1.3	Interface-3 (ATM ATM Appliance software – ATM driving APP service)	31
6.1.4	Interface-4 (ATM Driving APP service – Network Gateway INFRA)	31
6.1.5	Interface-5 (Network Gateway INFRA – Card Networks)	31
6.2	TECHNICAL CASH-IN Deposit PROCESS FLOW	32
6.2.1	Interface-1 (COD – ATM Appliance)	33
6.2.2	Interface-2 (ATM Appliance – NextGenATMIA APP)	33
6.2.3	Interface-3 (ATM ATM Appliance software – ATM driving APP service)	33
6.2.4	Interface-4 (ATM Driving APP service – Network Gateway INFRA)	33
6.2.5	Interface-5 (Network Gateway INFRA – Card Networks)	33
6.3	TECHNICAL CRYPTO-BUY PROCESS FLOW	34
6.3.1	Interface-1 (COD – ATM Appliance)	35
6.3.2	Interface-2 (ATM Appliance – NextGenATMIA APP)	35
6.3.3	Interface-3 (ATM ATM Appliance software – ATM driving APP service)	35
6.3.4	Interface-4 (ATM Driving APP service – Network Gateway INFRA)	35
6.3.5	Interface-5 (Network Gateway INFRA – Crypto Provider)	35
6.4	TECHNICAL CRYPTO-SELL PROCESS FLOW	36
6.4.1	Interface-1 (COD – ATM Appliance)	37
6.4.2	Interface-2 (ATM Appliance – NextGenATMIA APP)	37
6.4.3	Interface-3 (ATM ATM Appliance software – ATM driving APP service)	37
6.4.4	Interface-4 (ATM Driving APP service – Network Gateway INFRA)	37
6.4.5	Interface-5 (Network Gateway INFRA – Crypto Provider)	37



- 7 CERTIFICATION Specifications..... 38
 - 7.1 Certification Approach 39
 - 7.2 CERTIFICATION PROCESS FLOW 40
 - 7.2.1 Industry Certifications (Level-0 + Level-1) 40
 - 7.2.2 NextGenATMIA Certifications (Level-2) 41
 - 7.2.3 NextGenATMIA Certifications (Level-3) 41
 - 7.2.4 NextGenATMIA Certifications (Level-4) 41
 - 7.2.5 End-2-End NextGenATMIA Channel Certification 41
 - 7.3 Pre-Certification Registration 42
 - 7.4 Self-Certification Purpose 43
 - 7.5 Self-Certification Process 44
 - 7.5.1 Step-1: Apply 44
 - 7.5.2 Step-1: Execute 44
 - 7.5.3 Step-3 Validate 44
 - 7.5.4 Step-3 Register & Publish 44
 - 7.6 APPLY (Pre-Order) Self-Certification Levels 45

- APPENDIX A: HIGH-LEVEL ATM-APP API (ECMAScript 6) 46
- 8 Use Case Framework..... 47
 - 8.1 Version 1.0 48
 - 8.1.1 SERVICE use case 48
 - 8.1.2 CARD-AUTHENTICATION use case 48
 - 8.1.3 INPUT Cash-Out 49
 - 8.1.4 PROCESS Cash-Out 49
 - 8.1.5 OUTPUT Cash-Out 49
 - 8.2 Version 2.0 – CASH-IN (Deposit) 50
 - 8.2.1 INPUT Cash-IN 50
 - 8.2.2 PROCESS Cash-IN 50
 - 8.2.3 OUTPUT Cash-IN 50
 - 8.3 Version 2.0 - Crypto Currency BUY 51
 - 8.3.1 INPUT Crypto-BUY 51
 - 8.3.2 PROCESS Crypto-BUY 51
 - 8.3.3 OUTPUT CRYPTO-BUY 51
 - 8.4 Version 2.0 - Crypto Currency SELL 52
 - 8.4.1 INPUT CRYPTO-SELL 52
 - 8.4.2 PROCESS CRYPTO-SELL 52
 - 8.4.3 OUTPUT CRYPTO-SELL 52



- 9 JavaScript SPECIFICATIONS (2019, version 1.0) 53
 - 9.1 NextGen Object..... 53
 - 9.1.1 Get Service 53
 - 9.2 Session Service API..... 54
 - 9.2.1 Get Session Information 54
 - 9.2.2 Start In Service 55
 - 9.2.3 Start Session..... 56
 - 9.2.4 End Session 56
 - 9.2.5 Flows 57
 - 9.3 CardReader Service API..... 58
 - 9.3.1 Accept Card 58
 - 9.3.2 Eject Card 59
 - 9.3.3 Flows 60
 - 9.4 Card Service API 61
 - 9.4.1 Get Info 61
 - 9.4.2 Select Application..... 62
 - 9.5 PIN Service API 63
 - 9.5.1 Collect PIN 63
 - 9.5.2 Flow 64
 - 9.6 QRCode Reader Service API 65
 - 9.6.1 Get Info 65
 - 9.7 Withdrawal Service API..... 66
 - 9.7.1 Get Withdrawal Information..... 66
 - 9.7.2 Authorize..... 67
 - 9.7.1 Dispense..... 68
 - 9.7.2 Flow 69
 - 9.8 Receipt Service..... 70
 - 9.8.1 Print Receipt..... 70
 - 9.8.2 Flow 71
 - 9.8.3 Error Format..... 71



- 10 JavaScript SPECIFICATIONS (2020, version 2.0) 72
 - 10.1 NextGen Object..... 72
 - 10.2 Mixed Media Service API 73
 - 10.2.1 Get Mixed Media Information 73
 - 10.2.2 Start..... 74
 - 10.2.3 Cancel..... 75
 - 10.2.4 Collect Media 76
 - 10.2.5 Stop Collecting 76
 - 10.2.6 Capture Media 76
 - 10.2.7 Retain Media 77
 - 10.3 Deposit Service API 78
 - 10.3.1 Get Deposit Information 78
 - 10.3.2 Authorize..... 78
 - 10.3.3 Deposit 79
 - 10.3.4 MixedMedia and Deposit Service flow 80
 - 10.4 CryptoSell Service API 81
 - 10.4.1 Get CryptoSell Information 81
 - 10.4.2 Authorize..... 82
 - 10.4.3 Dispense..... 83
 - 10.5 One-Time Passcode Service API..... 83
 - 10.5.1 Collect OTP 83
 - 10.5.2 CryptoSell Flow 85
 - 10.6 CryptoBuy API 86
 - 10.6.1 Get CryptoBuy Information..... 86
 - 10.6.2 Authorize..... 86
 - 10.6.3 finalize 87
 - 10.6.4 CryptoBuy Flow 88

VERSION HISTORY

Version	Date	Author	Changes
1.0 FINAL	16-4-2020	M. Ficken	No additional comments received. Version approved by Steering Committee 23-4-2020.
1.1 Draft	7-6-2020	M. Ficken	New functionality based on ATMIA survey May 2020 added, see reviewcomment document 7-6-2020 for details.
1.6 draft	28-8-2020	M. Ficken	Process review comments FUNC-TECH-CERT working groups, see document 28-8-2020 for details and sync. version number with Blueprint 2020 review version number.
2.0 draft	30-10-2020	M.Ficken	Update Blueprint to 2020 version 2.0. Add Deposit (Cash-IN) Add Coin support (mixed media handling), Add Crypto Currency Buy and Sell functionality. Process review comments FUNC-TECH-CERT working groups.
2.0 Public Draft	2-11-2020	M.Ficken	Next-Gen Standards/Technical/Governance Committee approved Implementation Guide 2020 (incl. Blueprint 2020) to be published for 3 months public consultation

REFERENCES

No.	Document	Version	Author
1	Technical Standards Assessment Report	2.0 final	M. Ficken
2	Implementation Model	2.0	M. Ficken
3	NextGen Blueprint 2019	1.5, FINAL	ATMIA
4	Glossary of Terms, 2018-04	1	ATMIA
5	ATMIA anti-trust compliance policy – 10018816	Final	ATMIA
6	ATMIA IPR policy – 10020625	Draft	ATMIA
7	ATMIA Clarification Statement	2.0	ATMIA
8	NextGen ATM User Interface Best Practices	2018-10	ATMIA
9	NextGen Ecosystem Security for API App ATMs	2018-09	ATMIA
10	Added Value Services to boost Next Gen ATMs	2018-10	ATMIA
11	Participate in the NextGenATM Proof of Concept Q4 2019 till Q1 2020	18-9-2019	M. Ficken
12	First Next-Gen ATM Proof of Concept Results	March 2020	M. Ficken
13	Survey Results Next-Gen ATM Functionality 2020	May 2020	ATMIA
14	Next-Gen Blueprint 2020 version	2.0 draft	ATMIA

2 Introduction

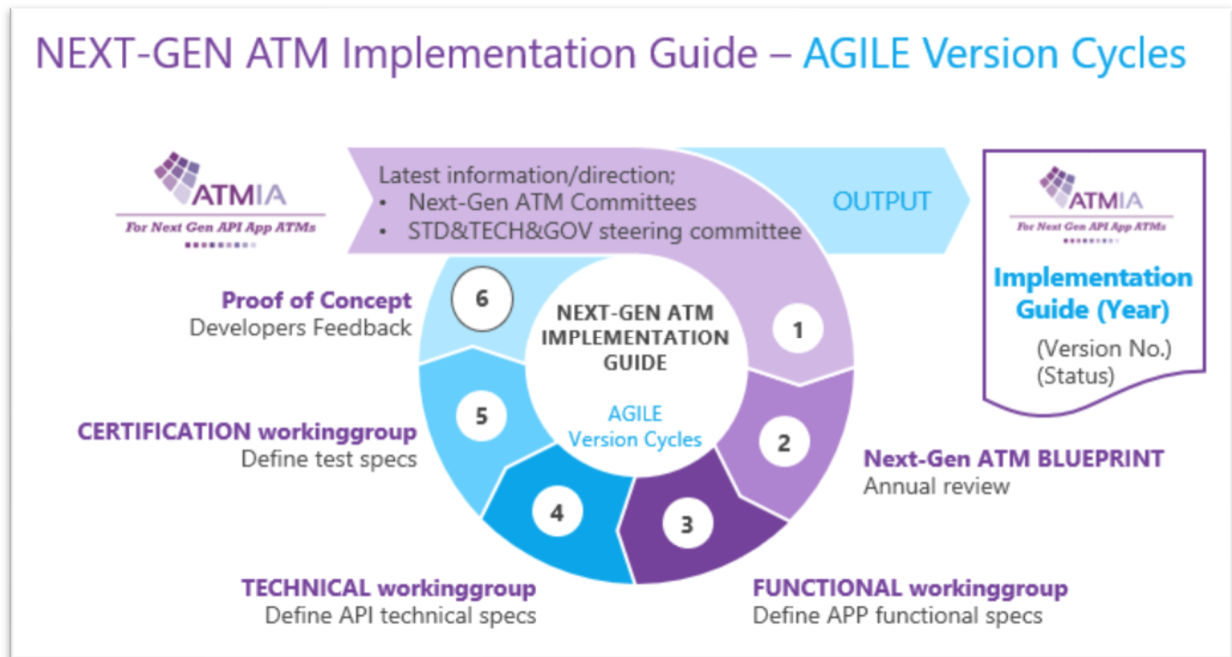
The Implementation Guide will contain the NextGenATMIA Specifications from Functional-, Technical and Certification perspective as a single source for NextGenATMIA providers to develop, test and certification of NextGen software.

The Technical Standards Roadmap (see reference 1) contains of the following three phases;

Phase-1	Phase-2	Phase-3
Define NextGenATMIA Implementation Guide 1.0: Containing NextGen Blueprint, Technical Standards, Vendor Agnostic Functional Flow (see paragraph 5.1) and Implementation Model (see paragraph 5.2)		
	Proof of Concept Implementations (see paragraph 5.4): to validate the specifications, gain implementation experience and demonstrate implementation reference showcases for ATMIA events in 2020.	
		NextGenATMIA Implementation Guide version 2.0: based on validated reference implementations to have the majority of possible NextGen implementation issues tackled

3 Approach

The approach for used for the implementation guide is following an Agile way of working. Each cycle version is based on a single NextGen APP functionality covering the end-2-end vendor agnostic interfaces in the NextGen Architecture Blueprint (NGA, see reference 3).



We realised the workgroup aim to finish phase-1 (implementation guide version 1.0) and have started phase-2 (Proof of Concept) at the end of 2019.

The review comments of the 3 months public consultation (December 2019 till March 2020) and the first Proof of Concept results review and improvements are processed in this final version too.

The first FINAL version 1.0 of the Implementation Guide 2019 is released at April 16 2020.

This document will result in the next version 2.0 of the Implementation Guide 2020 at the end of this year.


3.1 Implementation Committee

The mission of the Implementation Committee is;

Working together with industry stakeholders and standards bodies, the Next Gen ATM Implementation Committee will take all steps required to implement the standards roadmap and implementation model on behalf of the Standards & Technical Subcommittee, while managing and tracking the on-going progress of the implementation guide and reference implementations during 2019.

It's recommended that the reference implementations will focus on the ATM channel (ATM+COD Layer), especially the NextGen ATM APP. The other layers can rely on the ISO20022 standard.

The following objective and way of working is used by the implementation committee and working groups to define the implementation guide;



Implementation Guide - Objective

The objective of the ATMIA NextGen Committee and Workinggroups are;

to create a NextGenATM Implementation Guide as a global ATM industry standard specification (functional-, technical and test specifications)

owned and managed by ATMIA (NextGen) for a free market ATM Level Playing Field.

ATMIA's legal counsel will ensure safeguards are put in place to protect the project against IPR and Antitrust claims.

The Committee and Workinggroups are all accountable to the Standards & Technical committee for final decisions to sign off our implementation plan.

3.2 FUNCTIONAL Working Group

ATM product managers, ATM functional designers, ATM architects from especially ATM deployers (Bank, IADS, PSP, Processors) designed together with the NextGen Blueprint architects the desired vendor-agnostic functional requirements for an open NextGenATM App Model.

The Functional Working Group defined the first NextGen ATM app functionality.

3.3 TECHNICAL Working Group

ATM interface/API experts, JavaScript API designers/developers and ATM software developers from ATM software vendors designed the technical specifications for the implementation guide based on the desired vendor-agnostic functionalities.

The Technical Working Group defined the first NextGen ATM app and api.

3.4 CERTIFICATION Working Group

ATM test/certification expertise from ATM test tool vendors and ATM test/certification service providers designed and developed a testplan and testscripts for the development of a NextGen Certification system.

The Certification Work Group developed the first NextGen Certification system to validate the first NextGen ATM app reference implementation (when available by the software providers)

3.5 Proof of Concept (PoC)

ATMIA stimulate that the implementation Guide specifications will be validated in a Proof of Concept (PoC), to get developers implementation feedback from the ATM industry on the Implementation Guide to improve the quality and clarity of the specifications.

The PoC implementation will not be part of the implementation guide (conform reference 7, clarification statement, no software) and stays at the individual PoC implementation company. However ATMIA stimulate to demonstrate these PoC implementations during ATMIA events.

Innovators and early adopters could participate in NextGen ATM POC implementations.

First Next-Gen ATM PoC (Device Layer, API)

The first Next-Gen ATM Proof of Concept were based on version 1.0 December 2019 and presented during a workshop at the ATMIA US 2020 event in Houston.

The first Next-Gen ATM Proof of Concept RESULTS include PoC use case for Next-Gen ATM Contact(less), Additional Security Feature and API testing

The results can be found as whitepaper at the ATMIA website

<https://www.atmia.com/whitepapers/first-next-gen-atm-proof-of-concept-results/2388/>

During 2020 a second Proof of Concept will be started for IAD Retail ATMs with PoC use cases which will focus on COD/Mobile- and Cardless ATM Next-Gen ATM Blueprint.

If you are interested to participate in this PoC please contact mike@atmia.com

3.6 Next-Gen ATM Journey

In case you do not know yet, the following Next-Gen ATM documents can support you in your NextGen ATM direction and decisions;

NextGen ATM Quickscan 2019

<https://www.atmia.com/whitepapers/next-gen-atm-quickscan-2019--marcel-ficken/2355/>

Business Economic Matrix

<https://www.atmia.com/news/atmia-launches-next-gen-atm-business-case-toolkit/6952/>

Deployers Guide

<https://www.atmia.com/education/purchase-reports/next-gen-atm-deployer-guide/>

Watch the recording of the ATMIA webinar

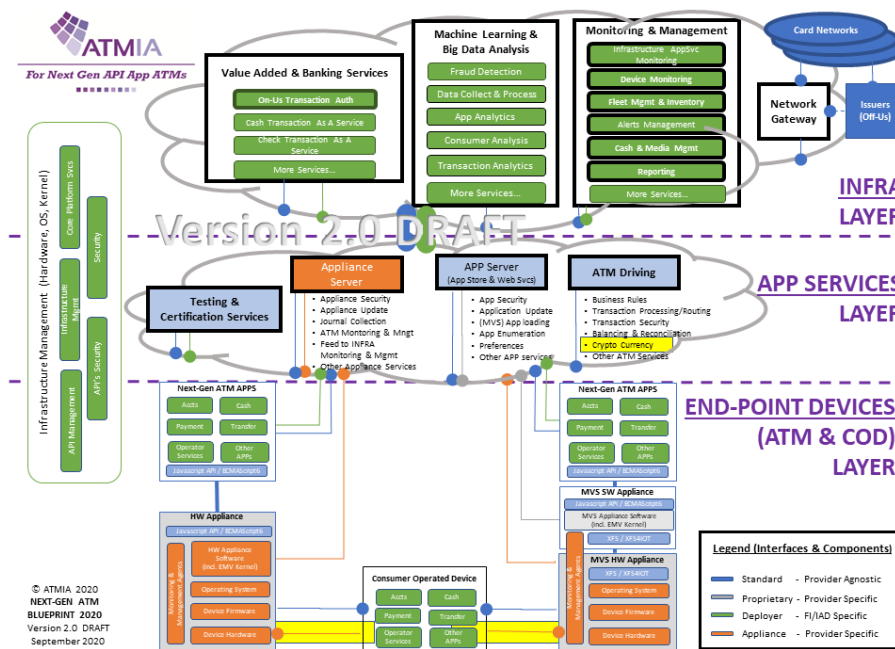
Next-Gen ATM journey to a CERTIFIED ATM Channel

<https://youtu.be/Du0MJqI-E9U>

4 NextGenATMIA Blueprint (version 2.0)

This Implementation Guide is a guideline for the implementation of the ATMIA Next Gen Blueprint (reference-3 and 2020 update reference 14), which contains three layers;

- INFRAstructure Layer
- APP services Layer
- End Point Devices Layer (ATM & Consumer Owned Devices)



This Next-Gen ATM Blueprint Architecture will be reviewed annually to be updated with the latest available technologies, architecture and implementation best practices in the market.

Implementation Guide 2019 version 1.0 were based on Next-Gen ATM Blueprint version 1.5 (reference-3). To get you up to speed see the following videoclip on the ATMIA Next-Gen portal "Next-Gen ATM Blueprint 2019 explained in 3 minutes";

<http://www.atmia.com/connections/members/atm-tube-videos/?video=lvtdXbFefUO>

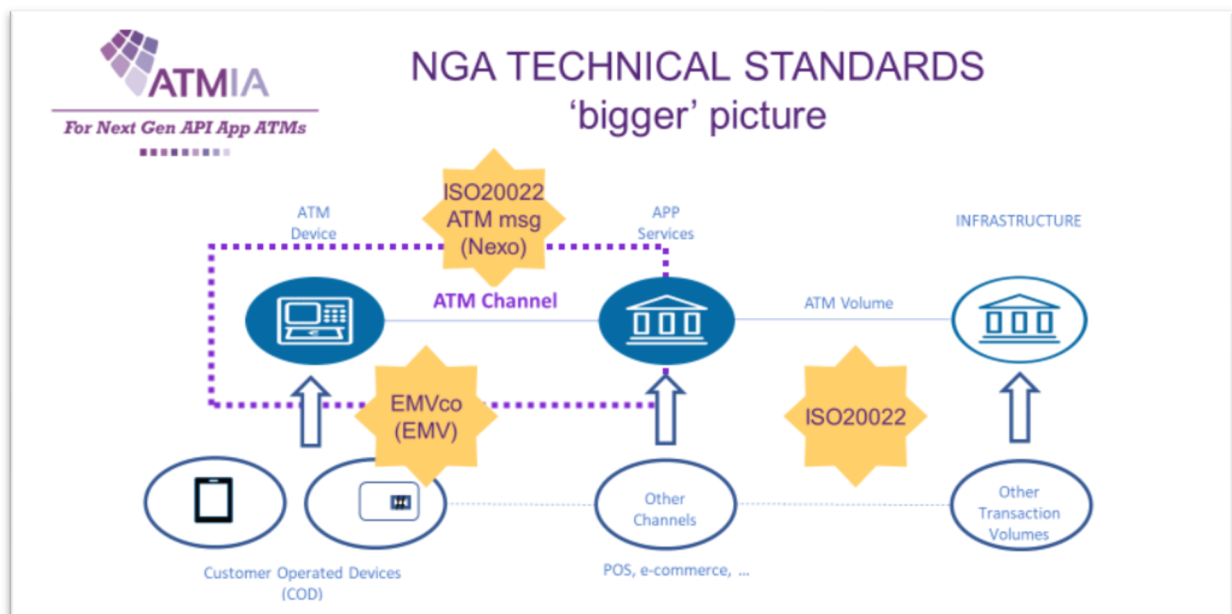
[This Implementation Guide 2020 version 2.0 is based on the annual reviewed Next-Gen ATM Blueprint 2020 version 2.0 \(reference 14\), see blueprint above.](#)

The public consultation of the Blueprint 2020 and Implementation Guide 2020 will be integrated.

4.1 Bigger Picture

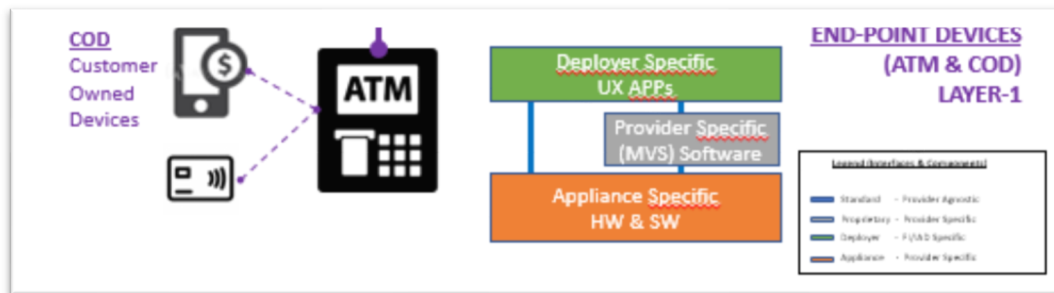
The ATM channel cannot be viewed in isolation. The ATM app services, and host infrastructure domains are part of a multi-channel environment that includes but is not limited to mobile (customer-owned devices or “CODs”) and POS channels. It is essential that this interactivity be taken into consideration when developing standards and aligning with current standards. Doing so will contribute to the efficiency and quality of implementation, support improved economies of scale, and help entities optimize their return on investment over the long term.

The ATM channel (ATM device and interfaces to the COD and APP Services) should be the primary focus for vendor agnostic standardisation, as shown in the figure below.



4.2 Blueprint – ATM end-point configurations

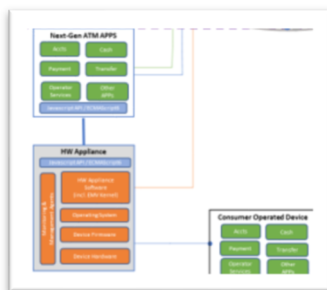
The Implementation Guide is using the ATMIA NextGen ATM Blueprint which defines two types of ATM end-point configurations (layer-1), which are explained in more detail in this paragraph.



The architecture basic of the ATM end-point device contain Appliance Provider Specific hardware and software shown in orange, can contain Provider Specific Multi-vendor software and other software shown in grey and Deployer Specific User Interface APP shown in green.

Which are connected through standard provider agnostic API's in blue, like JavaScript (ECMAScript6) for the high-end Deployer Specific APPs, and low-level XFS (or XFS4IOT) to connect to the ATM Appliance Provider Specific components.

Two basic ATM end-point Next-Gen configurations defined in the Next-Gen ATM Blueprint 2019 are;



High-Level ATM configuration

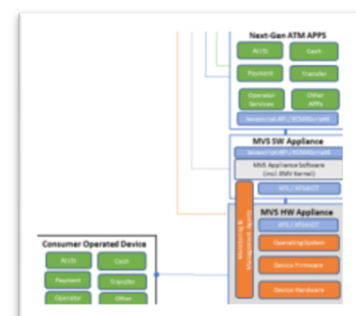
on the left in the Blueprint 2019 Device Layer.

Connecting the Deployer Specific ATM APPs (green) directly to the ATM Appliance Provider Specific hardware and software (orange).

MVS ATM configuration

on the right in the Blueprint 2019 Device Layer

Leverage on the current defacto MVS Multi-Vendor Software (in grey) and the current ATM appliances (in orange) using low-level XFS or next version XFS4IOT to connect the same high-level Deployer Specific ATM APPs (green).

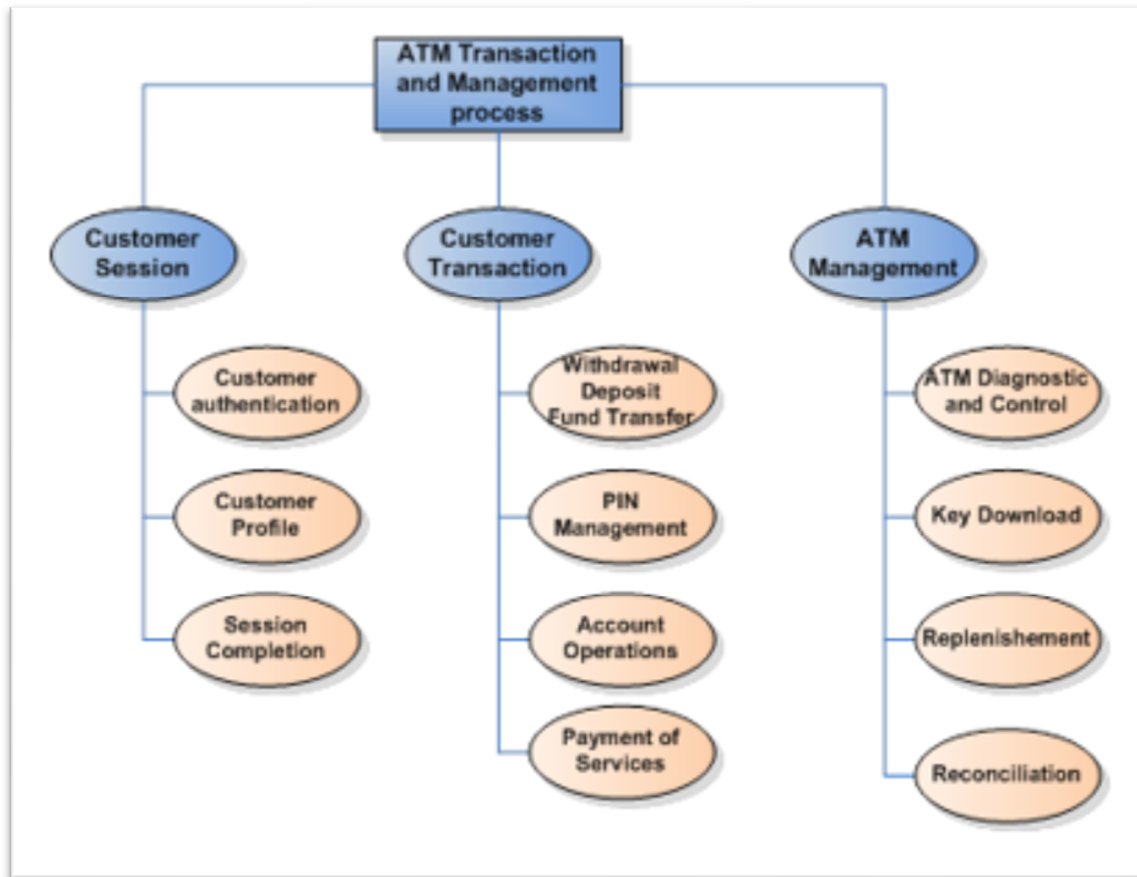


5 FUNCTIONAL Specifications

The Functional Use Cases chapter follows the NextGen Committees approved recommendations in the NextGen Standards Assessment Report (see reference 1) about NextGen App functionality.

The functionality in the NextGen Apps will be the main driving force and direction of the next steps. Starting with the NextGen App and transaction flow description of the NextGenATM Blueprint.

The high-level business processes covered by ISO 20022 ATM messages (Nexo) standard would be a good starting point, as shown in the figure below, which functionality should be provider agnostic.



Source: ISO20022 ATM message – high level business process

Recommendation: make the ATM user interface device-agnostic among ATMs or, even better, design the NGA App so the same NGA App can run on the ATM device and the COD.

FUNCTIONAL Overview

Overview of Vendor Agnostic APP-API Functionality;

Implementation Guide 2019, Version 1.0 FINAL

No	APP Vendor Agnostic		End-Point	
	Group	Function	ATM	COD
1	CASH	OUT (Magstripe, EMV contact)	X	N.A.
2	CASH	OUT (Contactless, NFC, QRCode,..)	X	X

Implementation Guide 2020, version 2.0 draft

No	APP Vendor Agnostic		End-Point	
	Group	Function	ATM	COD
3	CASH	IN Deposit (Contactless, NFC, QRCode.)	X	X
4	CASH (COD/Mobile)	Crypto Currency (Buy & Sell)	X	X

Future Version

No	APP Vendor Agnostic		End-Point	
	Group	Function	ATM	COD
5	Device Monitor Management	ATM Diagnose & Control	X	N.A.
6	PAYMENT	E-Commerce Purchase Cash Payment (deposit)	X	X
7	TRANSFER (COD/Mobile)	Instant Cash / Emergency Cash	X	X
8	CARDLESS (COD/Mobile)	Receive Customer Data from COD/Mobile (QRCode,) e.g. surcharge free ATM trxs by IADs	X	X
9	CASH (COD/Mobile)	Dynamic Currency Conversion (DCC)	X	X
	Value Added Services	Marketing Campaign See 50 examples to boost Next Gen ATMs (reference 10).		
	Accounts	Balance Inquiry		
	Other	???		

Out-of-Scope ATM/COD-APP functionality

1	FI/IAD specific	Customer User Interface	X	X
---	-----------------	-------------------------	---	---

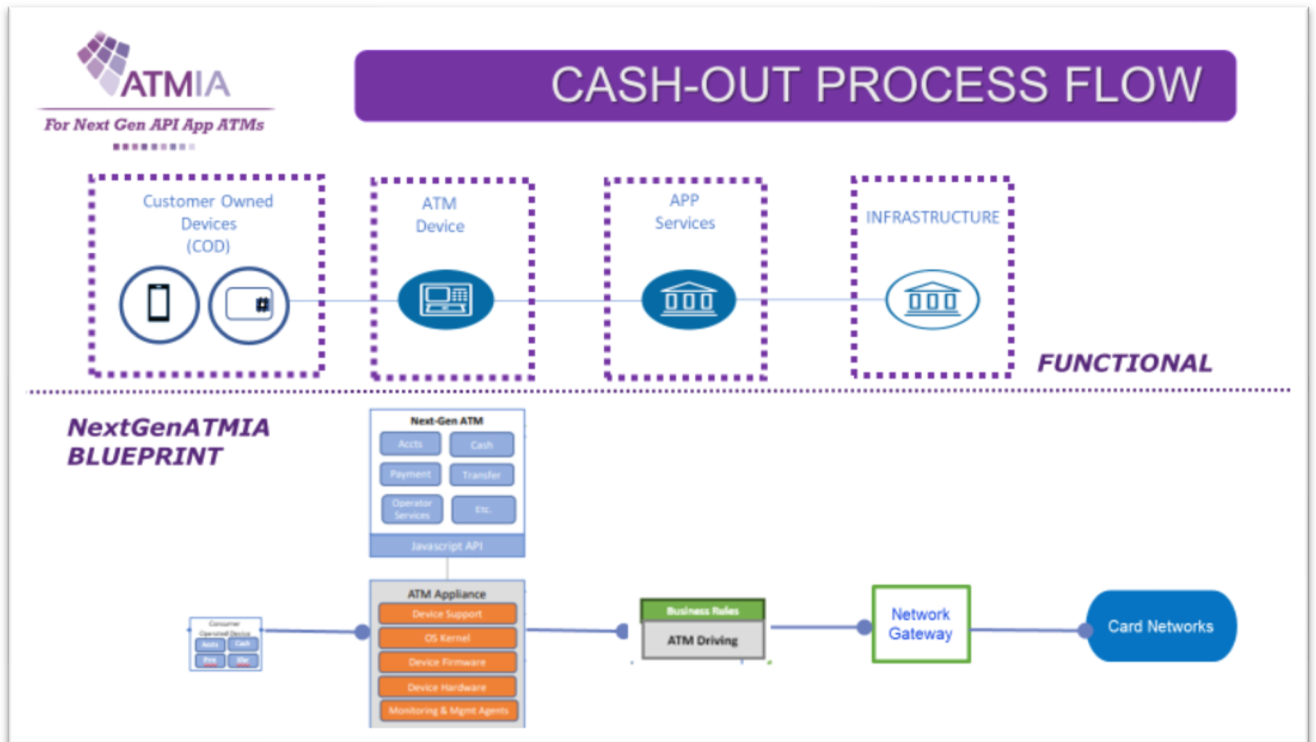
NON-FUNCTIONAL Overview

Overview of Vendor Agnostic Generic Non-Functional requirements.

No	Vendor Agnostic Non-Functional Requirements	
	Type	Requirement
1	Architecture	Support of multi-vendor NextGen ATM APPs
2		

5.1 CASH-OUT Functionality (version 1.0)

The first APP functionality will be the CASH-OUT Process Flow, which is drawn in the figure below in a FUNCTIONAL process flow based on the corresponding NextGenATMIA Blueprint process flow.



In the following paragraphs the vendor-agnostic CASH OUT functionality will be defined.

Customer User Interface is not vendor-agnostic and will be ATM deployer FI/IAD specific.

The Payment Scheme Specifications defines which functionality is supported too, especially for the endpoints and gateway to the card schemes.

Last but not least the ATMIA best practices for the User Interface and Security (see references 8+9).

The following Cash-Out Functional Transaction Types (Contact-Contactless-Cardless) have been defined in the table below.

Transaction type	User device / ATM interaction	User account identification	Credentials for authentication ①	Interoperability	Security Features
Contact Card Cash Withdrawal	Card / Card reader	Card PAN	PIN	Interoperable	
Contactless Card Cash Withdrawal	Card / NFC reader	Card PAN	PIN	Interoperable	
Contactless Cardbased Cash Withdrawal (COD wallet app)	COD / NFC reader	Card PAN	PIN	Interoperable using Host Card Emulation or Tokenization	Additional IoT Beacon Security Feature ②
PARTIAL Cardless (User Interaction) Cash Withdrawal	COD / ATM display screen (displaying QR code) ③	Bank's app login leading to virtualised card PAN	One Time Password +app authentication (touch ID or m-PIN)	Only Interoperable within Scheme (e.q.MasterCard "cardless" solution) (Need to be Tokenized - or Virtual PAN to process transaction)	Additional IoT Beacon Security Feature ②
FULL Cardless (End-to-End) Cash Withdrawal	COD / QR code reader ③ or keyboard entry	Bank's app or website login leading to customer id (pre-staged)	OTP + mobile PIN (m-PIN)	Only On-us or domestic (e.g. country scheme) (No Need for Tokenized - or Virtual PAN to process transaction)	Additional IoT Beacon Security Feature ②

① Supported Authentication method is choice of FI/deployer and issuer/scheme (including biometrics).

② Optional by Security Provider, see reference 12 (whitepaper First Next-Gen ATM PoC results, security use case)

③ Leverage on the EMVco QRCode specifications.

5.1.1 COD functionality

Customer User Interface is not vendor-agnostic and will be ATM deployer FI/IAD specific. The Payment Scheme Specifications defines which functionality is supported too, especially for the endpoints and gateway to the card schemes.

No.	Supported Functionality	Customer Owned Device
1	Customer Token / Technology	NFC / Contactless QRCode
2	Customer Authentication	PINcode Biometrics (optional)
3	Bank Note	Selection of notes (optional)
4	Coin Support	Mixed Media handling since version 2.0

5.1.2 ATM Device functionality

Customer User Interface is not vendor-agnostic and will be ATM deployer FI/IAD specific. The Payment Scheme Specifications defines which functionality is supported too, especially for the endpoints and gateway to the card schemes.

No.	Supported Functionality	NextGenATMIA ATM APP
1	Customer Token / Technology	Card Magstripe Card EMV ICC Contact Card EMV ICC Contactless QRCode
2	Customer Authentication	PINcode Biometrics (optional)
3	Bank Note	Selection of notes (optional)
4	Coins Support	Mixed Media handling since version 2.0

5.1.3 APP Service functionality

Most APP Services are generic for all the ATM DEVICE APP functionality and will be described in another common chapter.

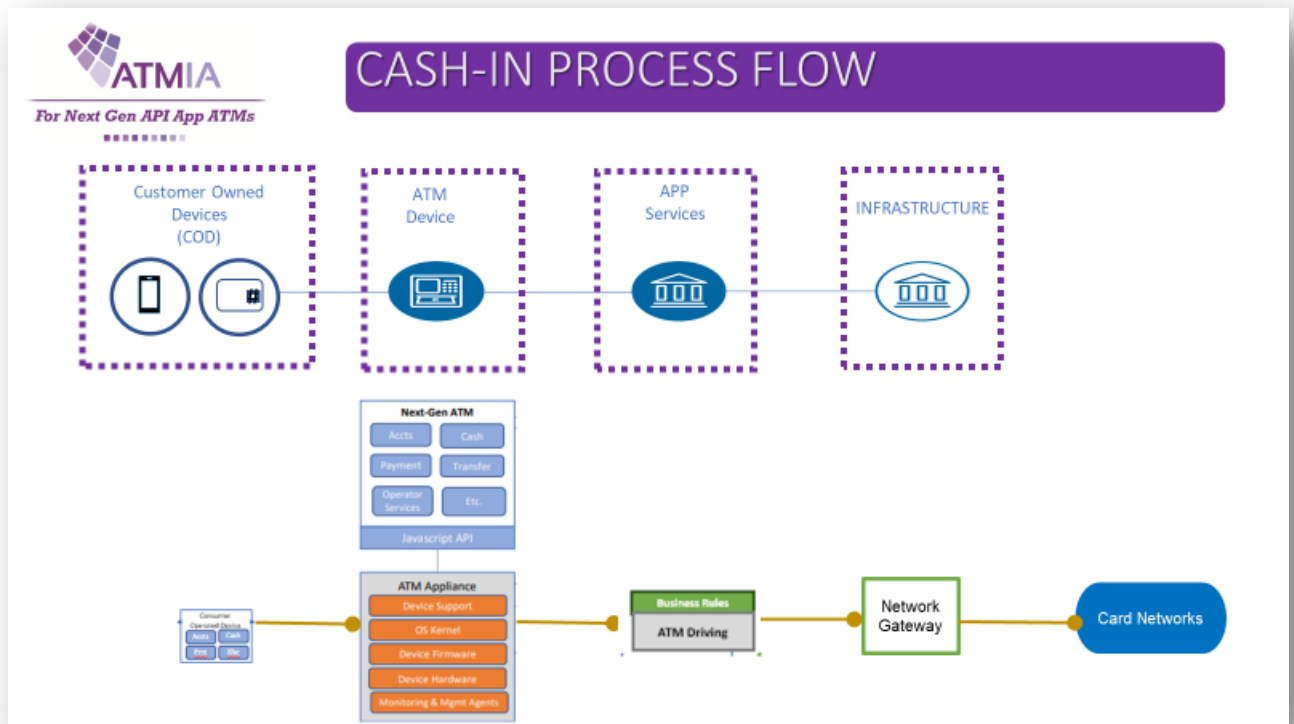
No.	APP Service	Functionality
1	ATM Driving Service	Provider Agnostic
2	ATM APP Service	Provider Specific
3	ATM APPLIANCE Service	Provider Specific
4	Testing & Certification Service	See Chapter 7.

5.1.4 INFRAstructure functionality

The Payment Scheme Specifications defines which functionality is supported too, especially for the endpoints and gateway to the card schemes.

5.2 CASH-IN Deposit Functionality (version 2.0)

The 2nd APP functionality will be the CASH-IN DEPOSIT Process Flow, which contains the same end-to-end process as the previous CASH-OUT chapter, the main difference is that cash (banknotes and/or coins) will go IN the ATM instead of come out of the ATM, based on the corresponding NextGenATMIA Blueprint process flow.



Customer User Interface is not vendor-agnostic and will be ATM deployer FI/IAD specific.

The Payment Scheme Specifications defines which functionality is supported too, especially for the endpoints and gateway to the card schemes.

Last but not least the ATMIA best practices for the User Interface and Security (see references 8+9).

The following Cash-IN Deposit Functional Transaction Types (Contact-Contactless-Cardless) have been defined in the table below.

Transaction type	User device / ATM interaction	User account identification	Credentials for authentication ①	Interoperability	Security Features
Contact Card Cash Deposit	Card / Card reader	Card PAN	PIN	Interoperable	
Contactless Card Cash Deposit	Card / NFC reader	Card PAN	PIN	Interoperable	
Contactless Cardbased Cash Deposit (COD wallet app)	COD / NFC reader	Card PAN	PIN	Interoperable using Host Card Emulation or Tokenization	Additional IoT Beacon Security Feature ②
PARTIAL Cardless (User Interaction) Cash Deposit	COD / ATM display screen (displaying QR code) ③	Bank's app login leading to virtualised card PAN	One Time Password +app authentication (touch ID or m-PIN)	Only Interoperable within Scheme (e.g. MasterCard "cardless" solution) (Need to be Tokenized - or Virtual PAN to process transaction)	Additional IoT Beacon Security Feature ②
FULL Cardless (End-to-End) Cash Deposit	COD / QR code reader ③ or keyboard entry	Bank's app or website login leading to customer id (pre-staged)	OTP + mobile PIN (m-PIN)	Only On-us or domestic (e.g. country scheme) (No Need for Tokenized - or Virtual PAN to process transaction)	Additional IoT Beacon Security Feature ②

① Supported Authentication method is choice of FI/deployer and issuer/scheme (including biometrics).

② Optional by Security Provider, see reference 12 (whitepaper First Next-Gen ATM PoC results, security use case)

③ Leverage on the EMVco QRCode specifications.

5.2.1 COD functionality

Customer User Interface is not vendor-agnostic and will be ATM deployer FI/IAD specific. The Payment Scheme Specifications defines which functionality is supported too, especially for the endpoints and gateway to the card schemes.

No.	Supported Functionality	Customer Owned Device
1	Customer Token / Technology	NFC / Contactless QRCode
2	Customer Authentication	PINcode Biometrics (optional)
3	Bank Note & Coins	Selection of notes (optional)

5.2.2 ATM Device functionality

Customer User Interface is not vendor-agnostic and will be ATM deployer FI/IAD specific. The Payment Scheme Specifications defines which functionality is supported too, especially for the endpoints and gateway to the card schemes.

No.	Supported Functionality	NextGenATMIA ATM APP
1	Customer Token / Technology	Card Magstripe Card EMV ICC Contact Card EMV ICC Contactless QRCode
2	Customer Authentication	PINcode Biometrics (optional)
3	Bank Note & Coins	Selection of notes (optional)

5.2.3 APP Service functionality

Most APP Services are generic for all the ATM DEVICE APP functionality and will be described in another common chapter.

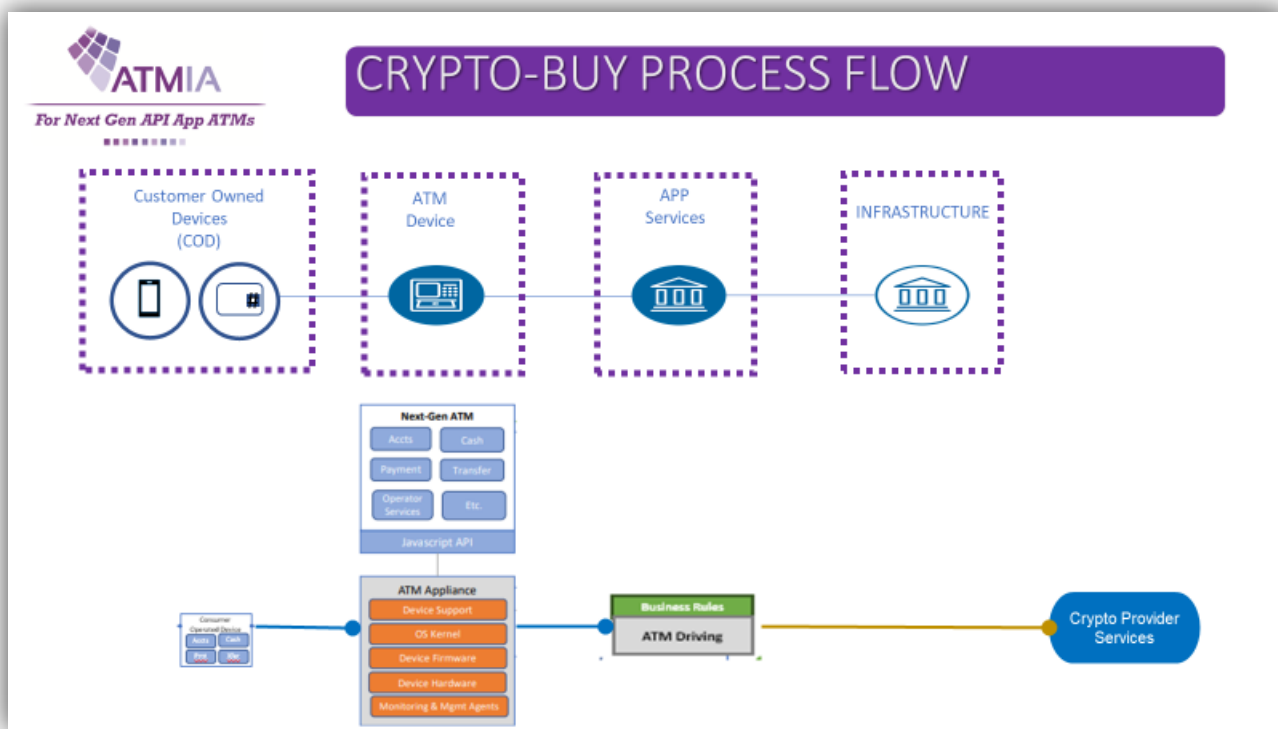
No.	APP Service	Functionality
1	ATM Driving Service	Provider Agnostic
2	ATM APP Service	Provider Specific
3	ATM APPLIANCE Service	Provider Specific
4	Testing & Certification Service	See Chapter 7.

5.2.4 INFRAstructure functionality

The Payment Scheme Specifications defines which functionality is supported too, especially for the endpoints and gateway to the card schemes.

5.3 CRYPTO CURRENCY – BUY Functionality (version 2.0)

The 3rd APP functionality will be the CRYPTO BUY Functionality Process Flow which follows the CASH-IN Deposit process flow, where the bank account card is replaced by the crypto currency wallet QRCode, based on the corresponding NextGenATMIA Blueprint process flow.



Customer User Interface is not vendor-agnostic and will be ATM deployer FI/IAD specific.

The Payment Scheme Specifications defines which functionality is supported too, especially for the endpoints and gateway to the card schemes.

Last but not least the ATMIA best practices for the User Interface and Security (see references 8+9).

The following CRYPTO-BUY Functional Transaction Types (Cardless) have been defined in the table below.

Transaction type	User device / ATM interaction	User account identification	Credentials for authentication ①	Interoperability	Security Features
FULL Cardless (End-to-End) Cash Crypto Buy	COD / QR code reader of Crypto Wallet	COD Crypto-Wallet)	COD Crypto Wallet)	Interoperable leverage on the crypto currency providers functionality	Additional IoT Beacon Security Feature ②

① Supported Authentication method is choice of FI/deployer and issuer/scheme (including biometrics).

② Optional by Security Provider, see reference 12 (whitepaper First Next-Gen ATM PoC results, security use case)

5.3.1 COD functionality

Customer User Interface is not vendor-agnostic and will be ATM deployer FI/IAD specific. The Crypto Currency Provider Specifications defines which functionality is supported too.

No.	Supported Functionality	Customer Owned Device
1	Customer Token / Technology	Crypto Wallet QRCode
2	Customer Authentication	Leverage on Crypto Wallet
3	Bank Note & Coins	Bank Note & optional coins too

5.3.2 ATM Device functionality

Customer User Interface is not vendor-agnostic and will be ATM deployer FI/IAD specific. The Payment Scheme Specifications defines which functionality is supported too, especially for the endpoints and gateway to the card schemes.

No.	Supported Functionality	NextGenATMIA ATM APP
1	Customer Token / Technology	QRCode Reader
2	Customer Authentication	N.A. Leverage on Crypto Wallet
3	Bank Note & Coins	Deposit Bank Notes & optional coins too

5.3.3 APP Service functionality

Most APP Services are generic for all the ATM DEVICE APP functionality and will be described in another common chapter.

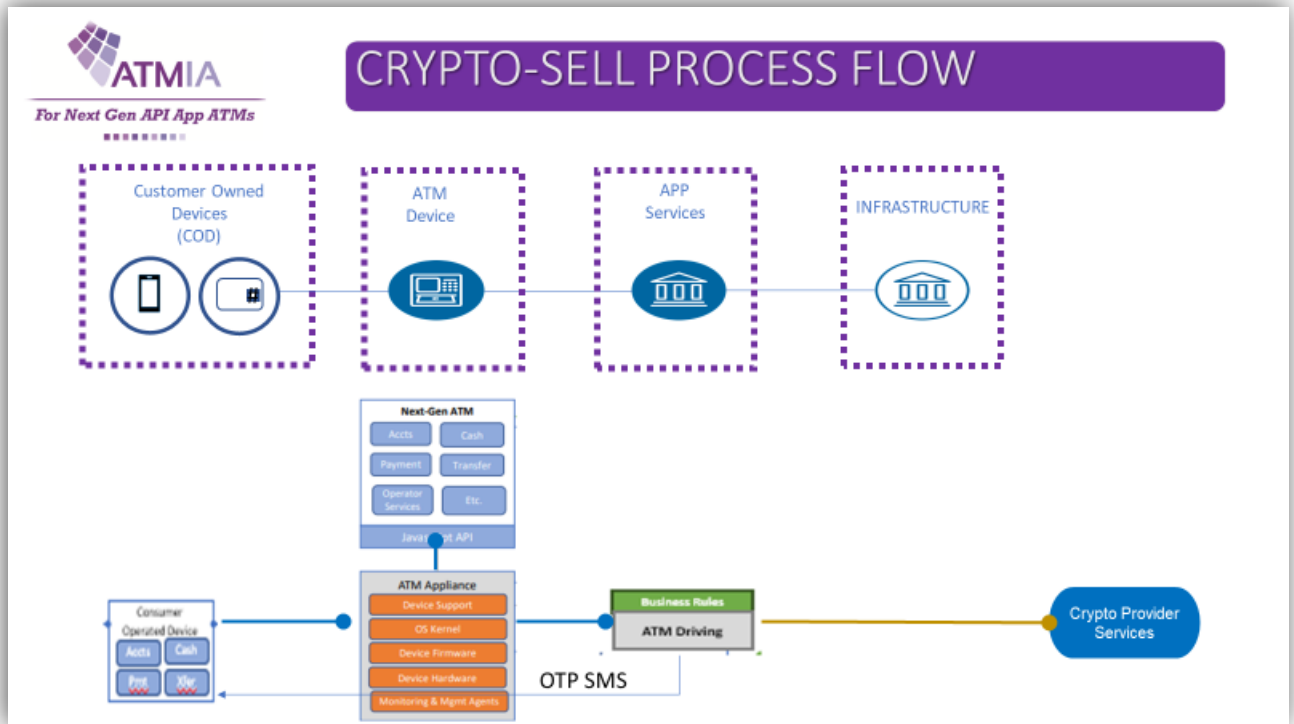
No.	APP Service	Functionality
1	ATM Driving Service	Provider Agnostic to End Point Layer, Provider Specific to Crypto Provider(s)
2	ATM APP Service	Provider Specific
3	ATM APPLIANCE Service	Provider Specific, Cash IN supported
4	Testing & Certification Service	See Chapter 7.

5.3.4 INFRAstructure functionality

The Payment Scheme Specifications defines which functionality is supported too, especially for the endpoints and gateway to the card schemes.

5.4 CRYPTO CURRENCY – SELL Functionality (version 2.0)

The 4th APP functionality will be the CRYPTO SELL Functionality Process Flow which follows the CASH-OUTDeposit process flow, where the bank account card is replaced by the crypto currency wallet QRCode, based on the corresponding NextGenATMIA Blueprint process flow.



Customer User Interface is not vendor-agnostic and will be ATM deployer FI/IAD specific.

The Payment Scheme Specifications defines which functionality is supported too, especially for the endpoints and gateway to the card schemes.

Last but not least the ATMIA best practices for the User Interface and Security (see references 8+9).

The following CRYPTO-SELL Functional Transaction Types (Cardless) have been defined in the table below.

Transaction type	User device / ATM interaction	User account identification	Credentials for authentication ①	Interoperability	Security Features
FULL Cardless (End-to-End) Cash Crypto Sell	COD / QR code reader of Crypto Wallet	COD Crypto-Wallet)	COD Crypto Wallet and One Time Password entered at ATM	Interoperable leverage on the crypto currency providers functionality	One-Time Password before Cash Out Additional IoT Beacon Security Feature ②

① Supported Authentication method is choice of ATM FI/deployer

② Optional by Security Provider, see reference 12 (whitepaper First Next-Gen ATM PoC results, security use case)

5.4.1 COD functionality

Customer User Interface is not vendor-agnostic and will be ATM deployer FI/IAD specific. The Crypto Currency Provider Specifications defines which functionality is supported too.

No.	Supported Functionality	Customer Owned Device
1	Customer Token / Technology	Crypto Wallet QRCode
2	Customer Authentication	Leverage on Crypto Wallet
3	Bank Note & Coins	Bank Note & optional coins too

5.4.2 ATM Device functionality

Customer User Interface is not vendor-agnostic and will be ATM deployer FI/IAD specific. The Payment Scheme Specifications defines which functionality is supported too, especially for the endpoints and gateway to the card schemes.

No.	Supported Functionality	NextGenATMIA ATM APP
1	Customer Token / Technology	QRCode Reader
2	Customer Authentication	Additional One Time Password (OTP) optional
3	Bank Note & Coins	Bank Notes & optional coins too

5.4.3 APP Service functionality

Most APP Services are generic for all the ATM DEVICE APP functionality and will be described in another common chapter.

No.	APP Service	Functionality
1	ATM Driving Service	Provider Agnostic to End Point Layer, Provider Specific to Crypto Provider(s)
2	ATM APP Service	Provider Specific
3	ATM APPLIANCE Service	Provider Specific
4	Testing & Certification Service	See Chapter 7.

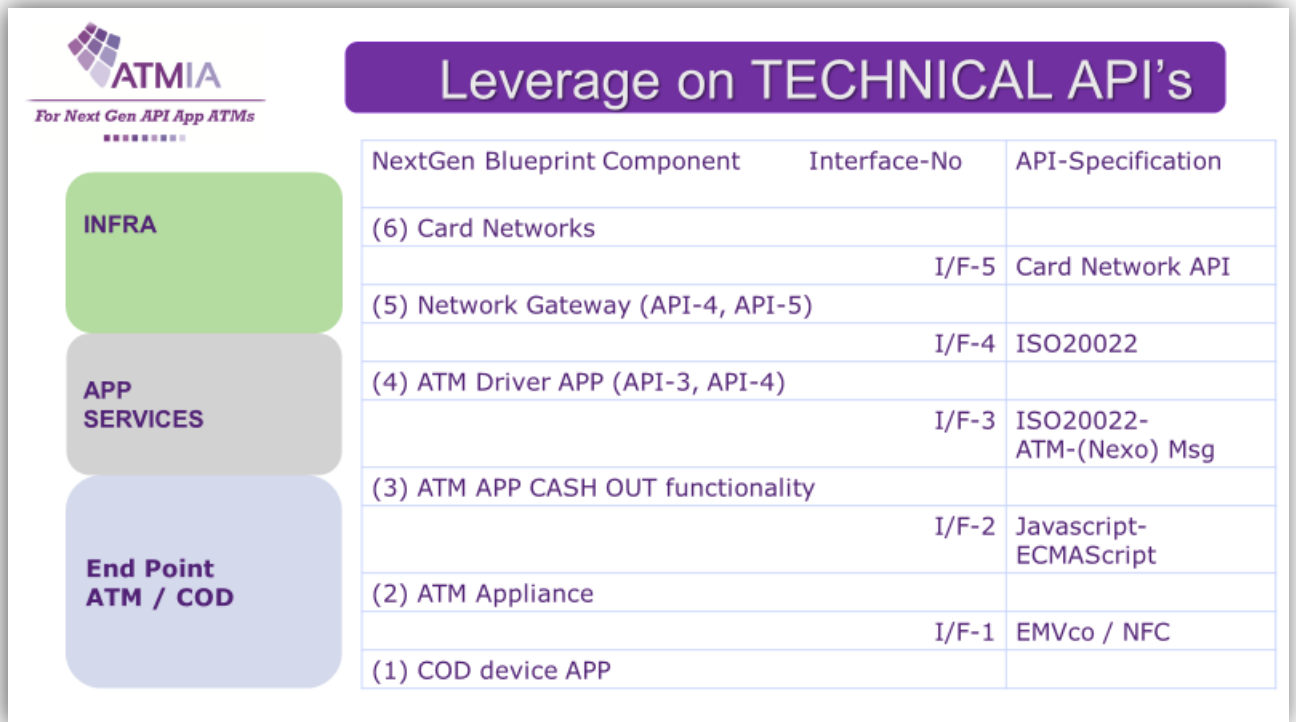
5.4.4 INFRAstructure functionality

The Payment Scheme Specifications defines which functionality is supported too, especially for the endpoints and gateway to the card schemes.

6 TECHNICAL Specifications

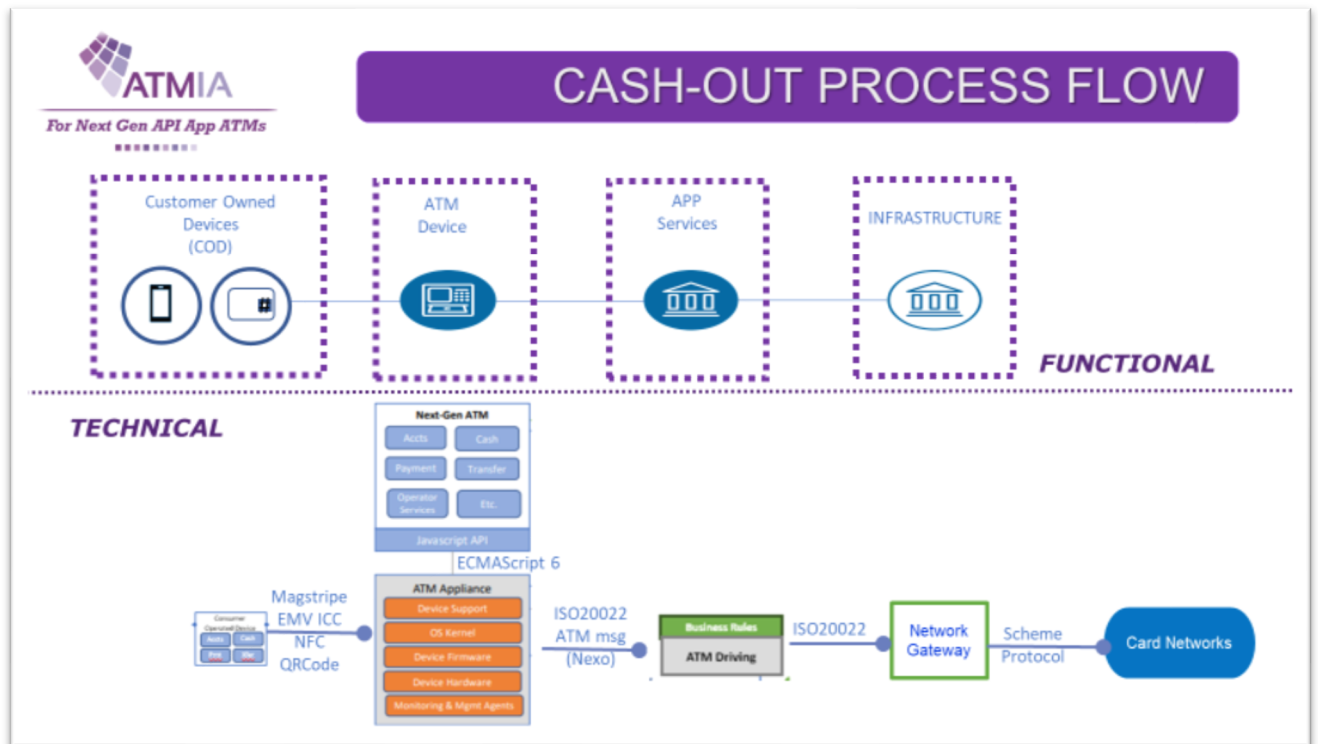
The NextGen Committees approved NextGen Standards Assessment Report (see reference 1) defines the following interface standards for the NextGen ATM App.

The following Technical Interface standards will be leveraged on;



6.1 TECHNICAL CASH-OUT PROCESS FLOW

The first APP functionality will be the CASH-OUT Process Flow, which is drawn in the figure below in a TECHNICAL process flow based on the corresponding FUNCTIONAL and NextGenATMIA blueprint process flow.



Customer User Interface is not vendor-agnostic and will be ATM deployer FI/IAD specific. The Payment Scheme Specifications defines which functionality and technology is supported too, especially for the endpoints and gateway to the card schemes.

6.1.1 Interface-1 (COD – ATM Appliance)

Customer User Interface is not vendor-agnostic and will be ATM deployer FI/IAD specific. The Payment Scheme Specifications defines which functionality and technology is supported too, especially for the endpoints by EMVco like Magstripe, EMV contact(less) and QRCode.

For Next-Gen ATM provider agnostic we will leverage on the EMVco QRCode for Payment Systems;

Merchant Presented Mode to be used by the deployer ATM-APP (display on the ATM screen)
Consumer Presented Mode to be used at the COD-APP screen and read by the ATM Appliance.

6.1.2 Interface-2 (ATM Appliance – NextGenATMIA APP)

A high-level Javascript ECMAScript6 API will be defined in Appendix A.

6.1.3 Interface-3 (ATM ATM Appliance software – ATM driving APP service)

Leverage on the ISO-20022 ATM Message (Nexo)

6.1.4 Interface-4 (ATM Driving APP service – Network Gateway INFRA)

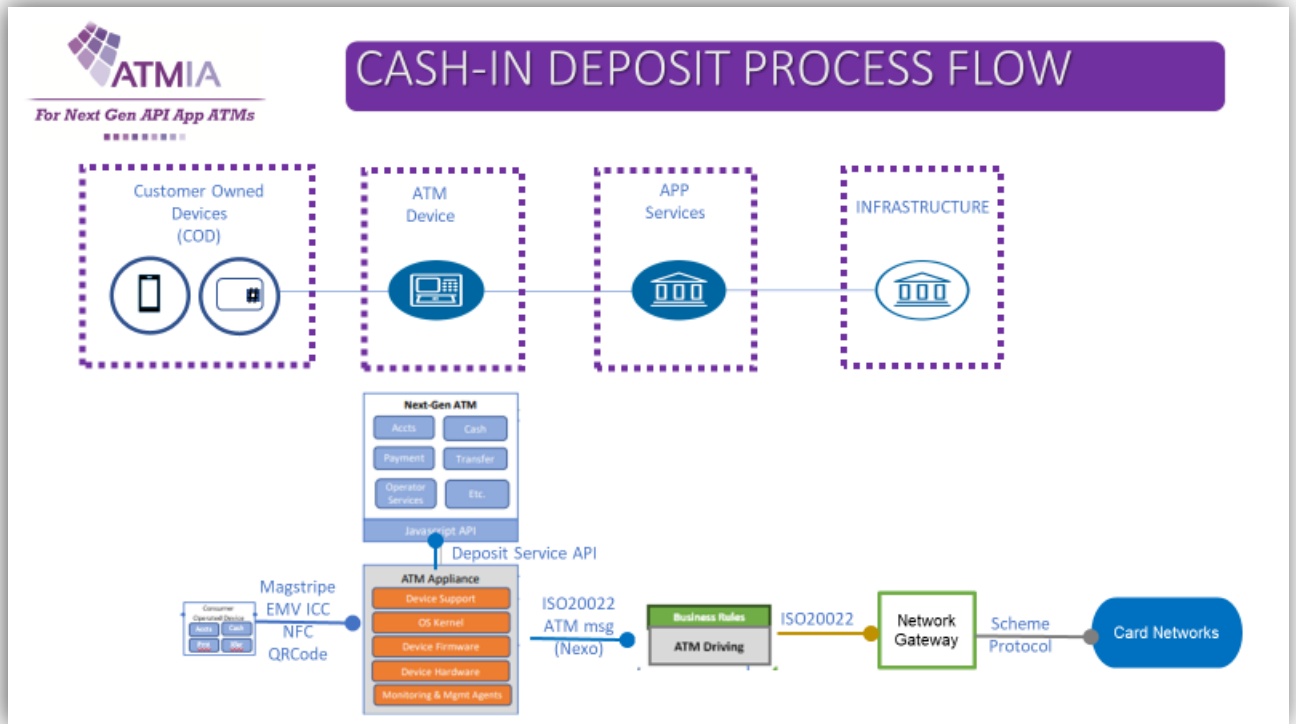
Leverage on the ISO-20022 Message

6.1.5 Interface-5 (Network Gateway INFRA – Card Networks)

Customer User Interface is not vendor-agnostic and will be ATM deployer FI/IAD specific. The Payment Scheme Specifications defines which functionality and technology is supported too, especially for the gateway to the card schemes like MasterCard- and VISA network specifications.

6.2 TECHNICAL CASH-IN Deposit PROCESS FLOW

The second APP functionality will be the CASH-IN Deposit Process Flow, which is drawn in the figure below in a TECHNICAL process flow based on the corresponding FUNCTIONAL and NextGenATMIA blueprint process flow.



Customer User Interface is not vendor-agnostic and will be ATM deployer FI/IAD specific. The Payment Scheme Specifications defines which functionality and technology is supported too, especially for the endpoints and gateway to the card schemes.

6.2.1 Interface-1 (COD – ATM Appliance)

Customer User Interface is not vendor-agnostic and will be ATM deployer FI/IAD specific. The Payment Scheme Specifications defines which functionality and technology is supported too, especially for the endpoints by EMVco like Magstripe, EMV contact(less) and QRCode.

For Next-Gen ATM provider agnostic we will leverage on the EMVco QRCode for Payment Systems;

Merchant Presented Mode to be used by the deployer ATM-APP (display on the ATM screen)
Consumer Presented Mode to be used at the COD-APP screen and read by the ATM Appliance.

6.2.2 Interface-2 (ATM Appliance – NextGenATMIA APP)

The high-level Javascript ECMAScript6 API will be defined in Appendix A. with the overall flow in chapter 8 and the High-Level ES6 **Deposit API** in chapter 10 version 2.0.

6.2.3 Interface-3 (ATM ATM Appliance software – ATM driving APP service)

Leverage on the ISO-20022 ATM Message (Nexo)

6.2.4 Interface-4 (ATM Driving APP service – Network Gateway INFRA)

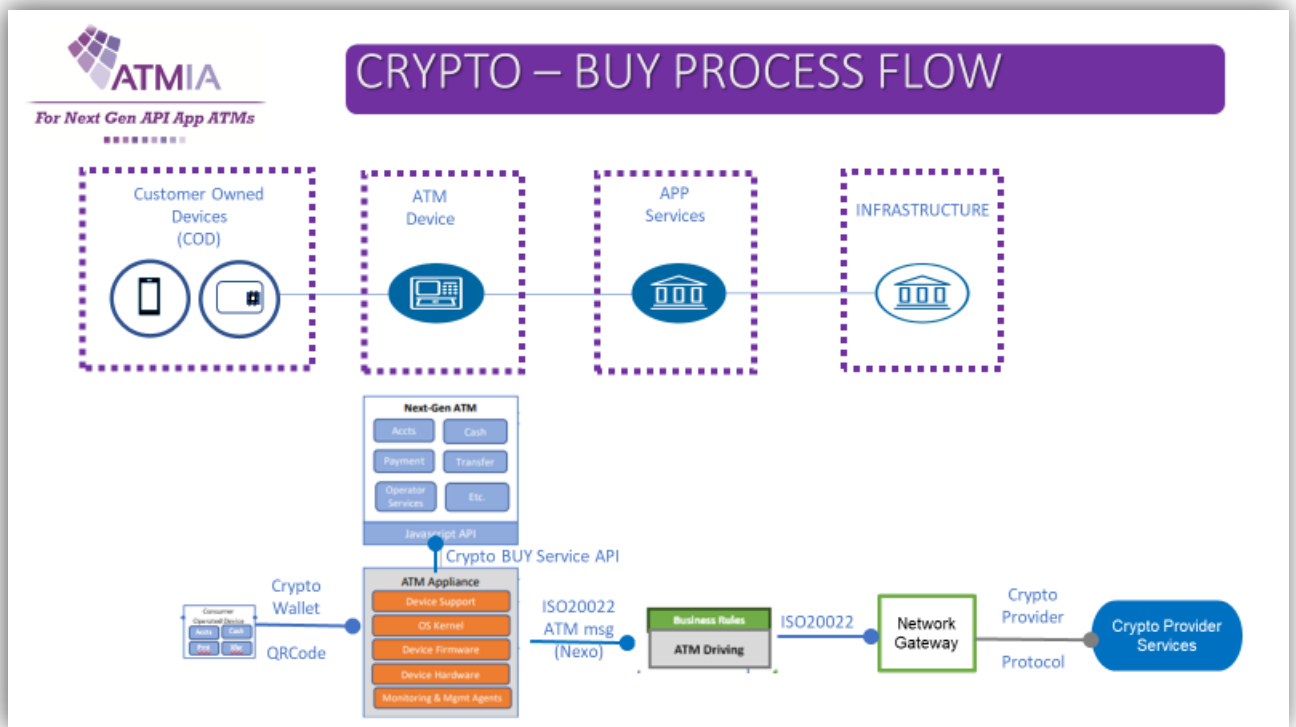
Leverage on the ISO-20022 Message

6.2.5 Interface-5 (Network Gateway INFRA – Card Networks)

Customer User Interface is not vendor-agnostic and will be ATM deployer FI/IAD specific. The Payment Scheme Specifications defines which functionality and technology is supported too, especially for the gateway to the card schemes like MasterCard- and VISA network specifications.

6.3 TECHNICAL CRYPTO-BUY PROCESS FLOW

The 3rd APP functionality will be the CRYPTO-BUY Process Flow, which is drawn in the figure below in a TECHNICAL process flow based on the Cash-IN Deposit and the corresponding FUNCTIONAL and NextGenATMIA blueprint process flow.



Customer User Interface is not vendor-agnostic and will be ATM deployer FI/IAD specific. The Crypto Currency Provider (provider specific) defines which functionality and technology is supported too, like the Crypto Currency Wallet QRCode.

6.3.1 Interface-1 (COD – ATM Appliance)

Customer User Interface is not vendor-agnostic and will be ATM deployer FI/IAD specific. The Crypto Currency Provider defines which functionality and technology is supported too. For Next-Gen ATM provider specific we will leverage on the Crypto Currency Wallet and QRCode.

6.3.2 Interface-2 (ATM Appliance – NextGenATMIA APP)

The high-level Javascript ECMAScript6 API will be defined in Appendix A. with the overall flow in chapter 8 and the High-Level ES6 **CryptoBUY API** in chapter 10 version 2.0.

6.3.3 Interface-3 (ATM ATM Appliance software – ATM driving APP service)

Leverage on the ISO-20022 ATM Message (Nexo)

6.3.4 Interface-4 (ATM Driving APP service – Network Gateway INFRA)

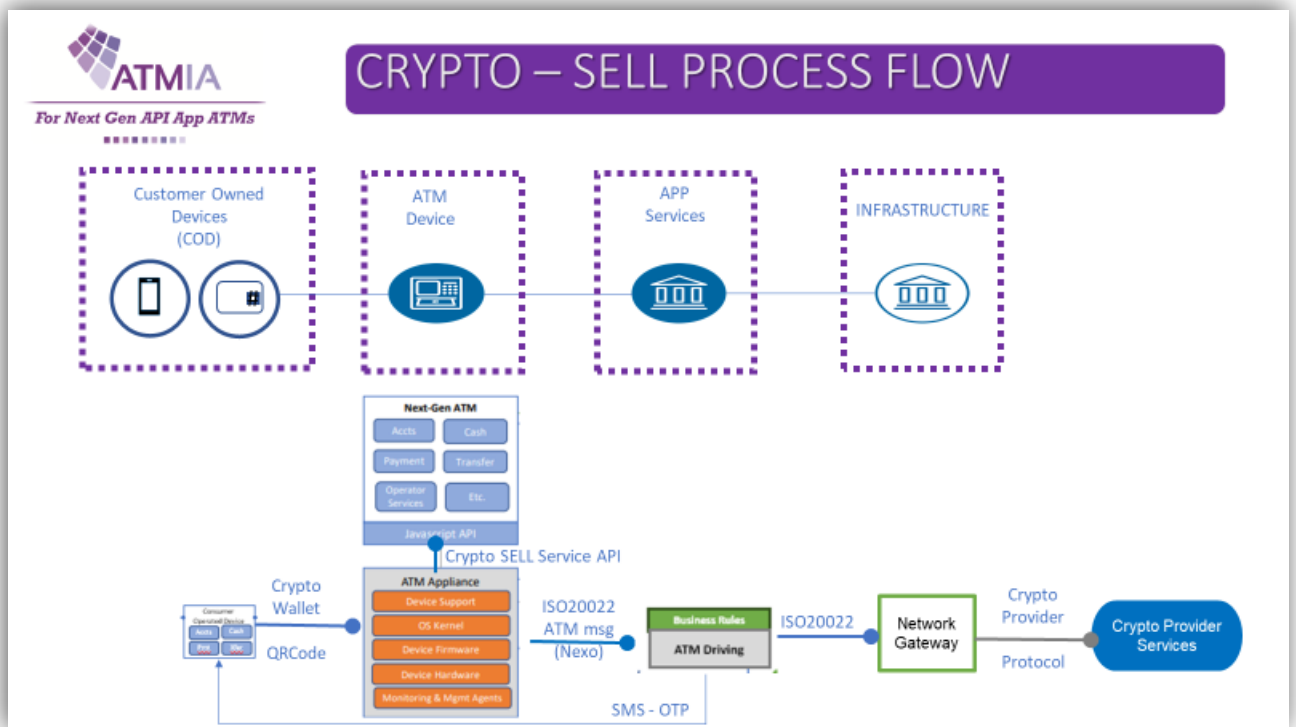
Leverage on the ISO-20022 Message

6.3.5 Interface-5 (Network Gateway INFRA – Crypto Provider)

The Crypto Currency Provider Specifications defines which functionality and technology is supported, for the gateway to the crypto currency provider.

6.4 TECHNICAL CRYPTO-SELL PROCESS FLOW

The 4th APP functionality will be the CRYPTO-SELL Process Flow, which is drawn in the figure below in a TECHNICAL process flow based on the Cash OUT and the corresponding FUNCTIONAL and NextGenATMIA blueprint process flow.



Customer User Interface is not vendor-agnostic and will be ATM deployer FI/IAD specific. The Crypto Currency Provider (provider specific) defines which functionality and technology is supported too, like the Crypto Currency Wallet QRCode.

6.4.1 Interface-1 (COD – ATM Appliance)

Customer User Interface is not vendor-agnostic and will be ATM deployer FI/IAD specific. The Crypto Currency Provider defines which functionality and technology is supported too. For Next-Gen ATM provider specific we will leverage on the Crypto Currency Wallet and QRCode.

6.4.2 Interface-2 (ATM Appliance – NextGenATMIA APP)

The high-level Javascript ECMAScript6 **CryptoSELL** API will be defined in Appendix A. with the overall flow in chapter 8 and the High-Level ES6 API in chapter 10 version 2.0. For additional security the phone number is entered at the ATM, to be able to receive a One Time Password (OTP) by SMS, which need to be entered at the ATM before dispensing the cash.

6.4.3 Interface-3 (ATM ATM Appliance software – ATM driving APP service)

Leverage on the ISO-20022 ATM Message (Nexo). For additional security an One Time Password / Validation Code is send by SMS to the provided phone number of the COD Crypto Currency Wallet device, which need to be entered at the ATM before dispensing the cash

6.4.4 Interface-4 (ATM Driving APP service – Network Gateway INFRA)

Leverage on the ISO-20022 Message

6.4.5 Interface-5 (Network Gateway INFRA – Crypto Provider)

The Crypto Currency Provider Specifications defines which functionality and technology is supported, for the gateway to the crypto currency provider.

7 CERTIFICATION Specifications

The following DRAFT mission of the Certification Working Group is defined;

To build an achievable, pro-competitive next gen certification system which takes account of available resources and which delivers a certification service designed to ensure global vendor interoperability and ubiquity of next gen ATMs, focusing primarily, but not exclusively, on the vendor-agnostic infrastructure level of the blueprint, whilst encouraging the widest possible industry participation and adoption.

The following Certification Levels are defined for NextGenATMIA;

NextGenATMIA CERTIFICATION LEVELS			
NextGen Blueprint	Certification Levels	Certification Level Description	
INFRASTRUCTURE	Level-4	NextGenATMIA INFRA-Service	
APP Services	Level-3	NextGenATMIA APP-Service	
End Point Devices ATM & COD	Level-2	NextGenATMIA ATM Appliance	NextGenATMIA ATM APP
	Level-1	ATM Appliance (or ATM App UX)	Customer Owned Device (COD)
Leverage on Pre-Conditions	Level-0	Payment Scheme Certifications (Acquirer, Network,..) PCI certification EMVco Certifications (Level-1, Level-2, ...)	

7.1 Certification Approach

The following Certification Approach is defined how to validate the functionality (chapter 5) and technical interfaces (chapter 6);

- **Validation of single NextGen Blueprint component**
- **BlackBox testing using Interfaces (API)**
- **Leverage on Certification Standards from payment schemes (MC, VISA), EMVco, PCI, ..**
- **Define NextGenATMIA Implementation Confirmation Statement (ICS) for modular certification per APP or API per Certification Level.**

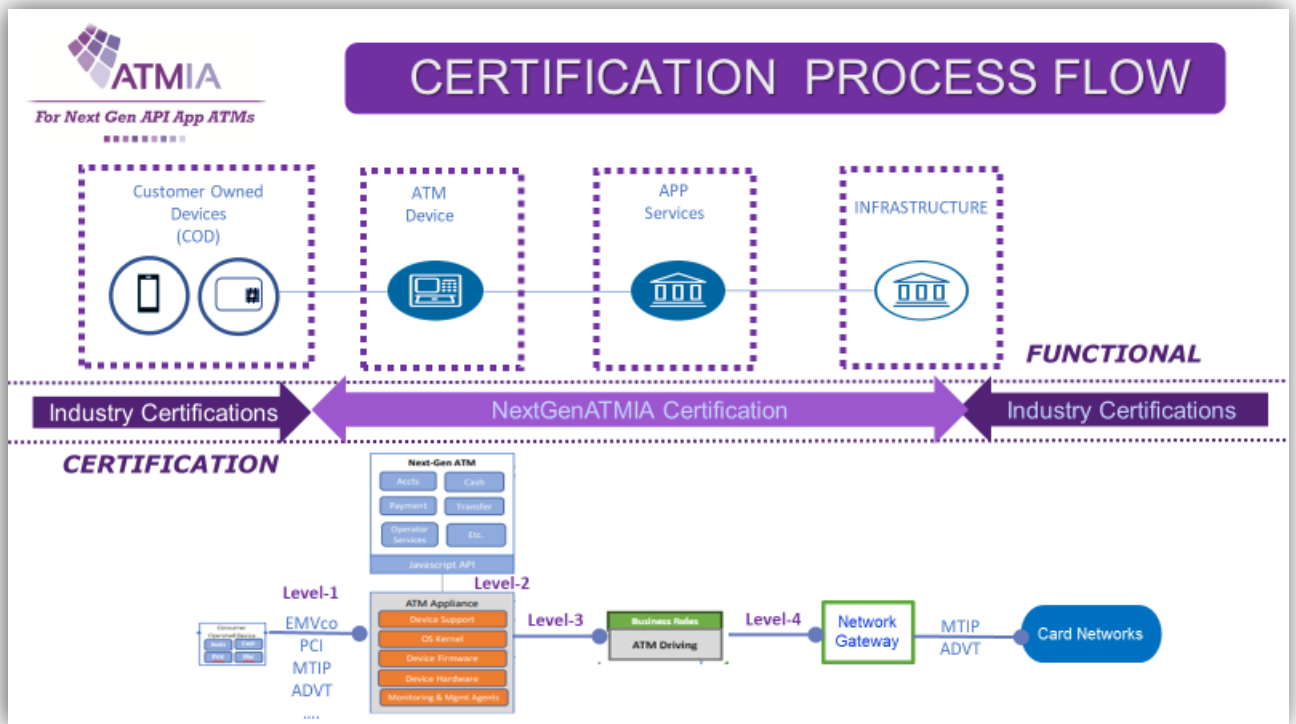
NextGenATMIA CERTIFICATION LEVELS				
NextGen Blueprint	Certification Levels	Certification Level Description		HOW to Validate ? (Blackbox – Interface)
INFRASTRUCTURE	Level-4	NextGenATMIA INFRA-Service		I/F-4 ISO20022
APP Services	Level-3	NextGenATMIA APP-Service		I/F-3 ISO20022
End Point Devices ATM & COD	Level-2	NextGenATMIA ATM Appliance	NextGenATMIA ATM APP	I/F-2 JavaScript ES6
	Level-1	ATM Appliance	NextGenATMIA COD-APP	I/F-1
		ATM-APP (UX)		
Leverage on Pre-Conditions	Level-0	Payment Scheme Certifications (Acquirer, Network,..) PCI certification EMVco Certifications (Level-1, Level-2, ...)		

7.2 CERTIFICATION PROCESS FLOW

The certification functionality will follow the different TECHNICAL Process Flows, which is drawn in the figure below in a CERTIFICATION process domains based on the corresponding FUNCTIONAL and TECHNICAL process flows.

7.2.1 Industry Certifications (Level-0 + Level-1)

NextGenATMIA Certification will leverage on the industry certifications from EMVco, PCI and the payment schemes terminal integration process and acquirer processor/network certifications for ATM devices, like shown in the figure.



EMVco Certifications

EMVCo Level-1 Hardware, Level-2 EMV kernel, ...

PCI Certifications

PCI-PED, PCI-DSS, PCI-...

MasterCard Certification

M-TIP for ATM

VISA Certifications

ADVT for ATM

7.2.2 NextGenATMIA Certifications (Level-2)

Software applications using the JavaScript ES6 API (chapter 8-10) between ATM Appliance and ATM APP.

7.2.3 NextGenATMIA Certifications (Level-3)

Software applications using the ISO20022 API between Layer-1 EndPoint Device and Layer-2 APP Services

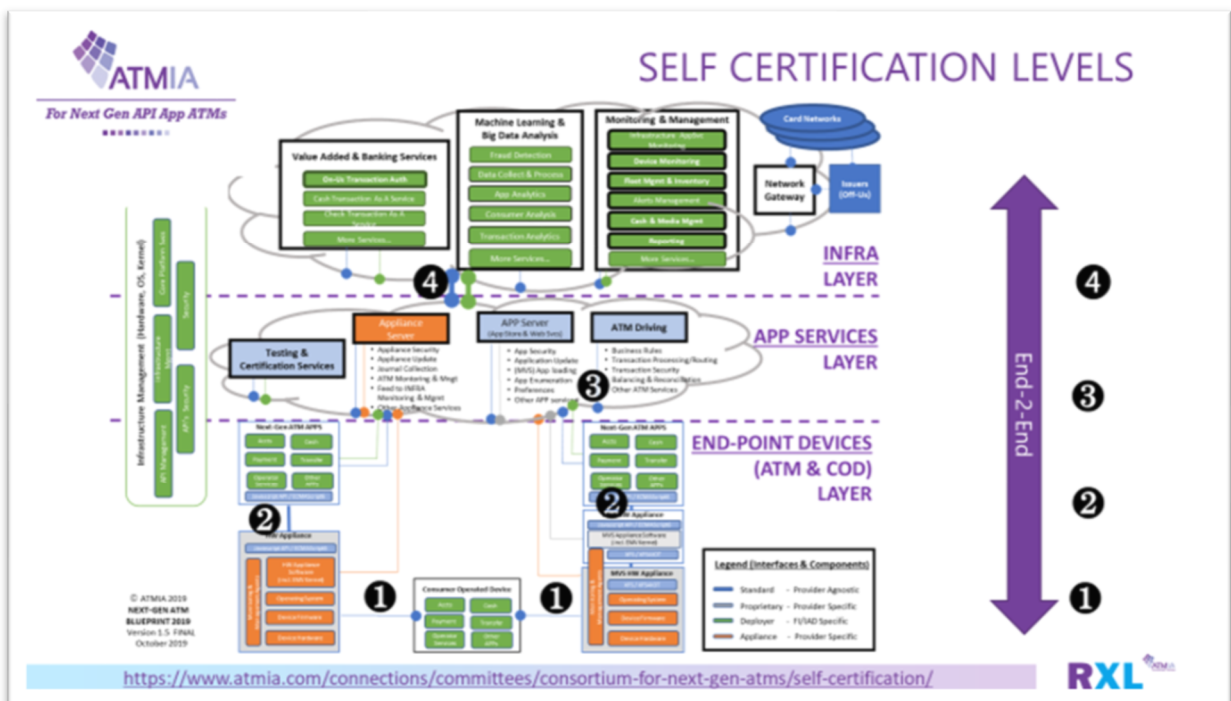
7.2.4 NextGenATMIA Certifications (Level-4)

Software applications using ISO20022 API between Layer-2 APP Services and Layer-3 INFRA Services

7.2.5 End-2-End NextGenATMIA Channel Certification

The chain of all NextGenATMIA Certification provider components Level-1 till Level-4 to provide the deployer a Next-Gen-ATMIA End-2-End ATM channel.

The figure below provide an overview of the different Self Certification Levels.



7.3 Pre-Certification Registration



Pre-register now if you want to be one of the first companies who can start the Next-Gen ATMIA Self-Certification Process in 2020, once it goes live later this year to certify your Next Gen products and services.


Introduced during ATMIA US 2020 event in the ATMIA event app and at the ATMIA Next-Gen portal <https://www.atmia.com/connections/committees/consortium-for-next-gen-atms/self-certification/>

7.4 Self-Certification Purpose

The purpose of Next-Gen ATM Self Certification can be defined as follows;

Independent Next-Gen ATM Functional API Assessment
of
YOUR Provider Agnostic Next-Gen ATM component

to support

Easy  Next-Gen ATM Integration
and
Gain YOUR Next-Gen ATM benefits
as
Next-Gen ATM Provider and Deployer

All stakeholders can benefit from Next-Gen ATM Self Certification at a Level Playing Field.

7.5 Self-Certification Process

The Self-Certification process contains of the following basic steps;

7.5.1 Step-1: Apply

The Provider or Deployer will apply for Self-Certification at the Certification Authority Board (CAB) and provide information about the Next-Gen ATM component including level-0 and ICS Level information to start the self-certification process.

7.5.2 Step-1: Execute

The CAB will send the provider the information needed to execute the Self Certification for the requested ICS Level. The provider execute the self-certification cycle.

7.5.3 Step-3 Validate

When the provider is finished the self-certification cycle will be closed.

The CAB Self Certification System will automatic validate the Self-Certification ICS Level Results. If not all tests are passed, a next cycle can be started.

7.5.4 Step-3 Register & Publish

When all self-certification level results are passed the CAB will confirm, register and publish the Next-Gen ATM ready Level at the ATMIA portal.

This will be an automated process. Only exceptions and disputes will be handled manually.

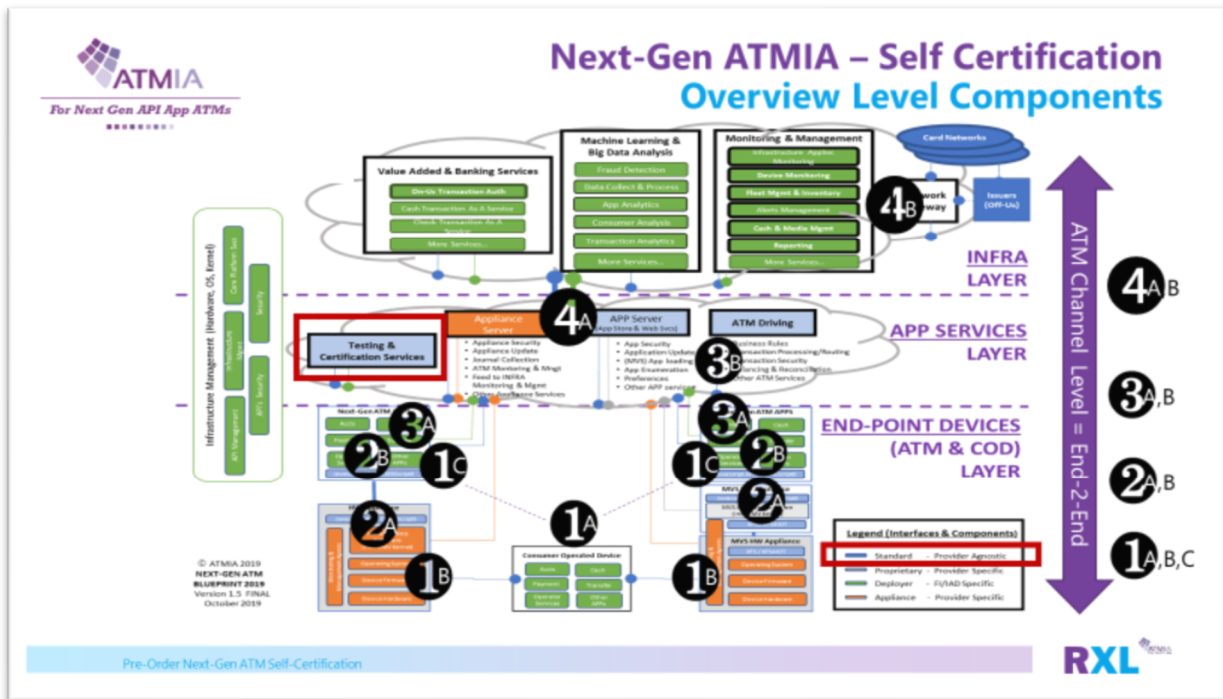
PROCESS	STEP-1	STEP-2	STEP-3	STEP-4
Request by	PROVIDER DEPLOYER	CAB	PROVIDER DEPLOYER	CAB
Activity	Apply Level 0 ICS Level x	Execute Self- Certification ICS Level	Validate Self-Certification ICS Level Results	Confirm, Register & Publish ICS level
Response by	CAB	PROVIDER	CAB	CAB
Result	Start Process	Provide Results End Process	Assessment Self-Certification ICS Level Results	Confirm to Provider Register Provider ICS Level Publish ATMIA portal

The operational implementation of each step will be defined, designed in the next chapters and implemented (developed and deployed).

7.6 APPLY (Pre-Order) Self-Certification Levels

During the ATMIA webinar: The Next-Gen ATM journey towards a CERTIFIED ATM Channel in August 2020, the certification process and pre-order is explained and launched at; <https://www.atmia.com/connections/committees/consortium-for-next-gen-atms/self-certification/>

Each software component with a provider-agnostic API (blue) can be Next-Gen ATMIA CERTIFIED !!



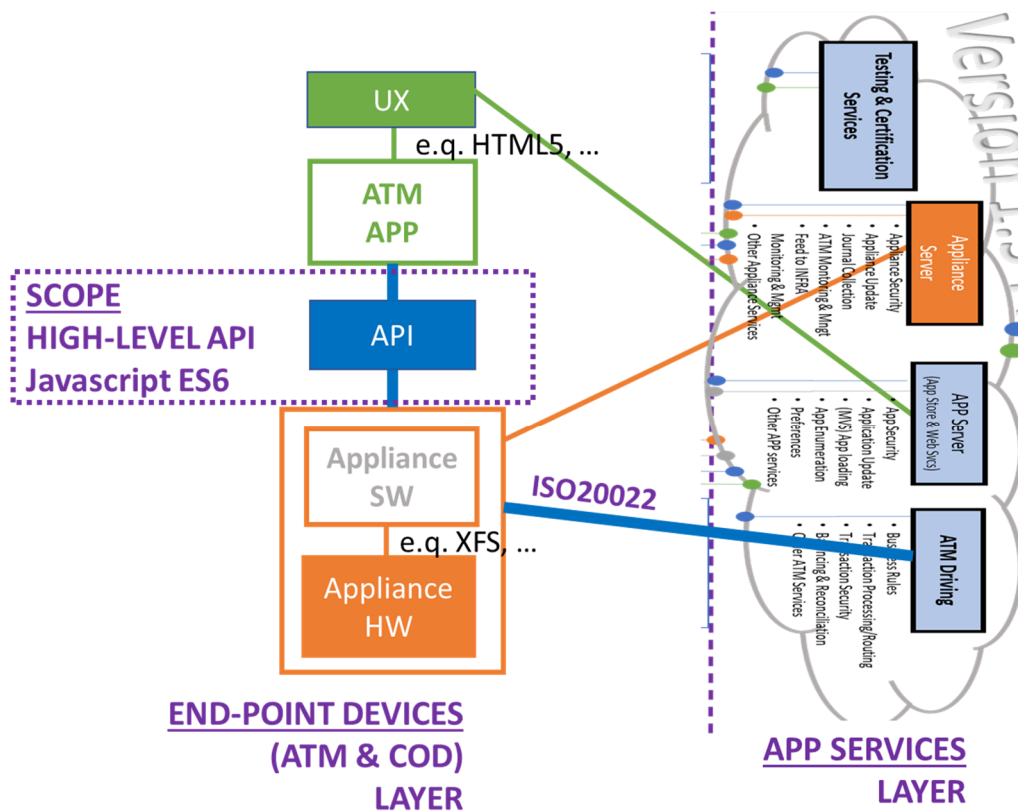
No	Blueprint Layer	Application Component	Level-1	Level-2	Level-3	Level-4
1A	End-Point	COD	COD			
1B	Devices	ATM Appliance	ATM Appl			
1C		ATM-APP	ATM APP			
2A	End-Point	ATM Appliance		ATM-Apl		
2B	Devices	ATM APP		ATM-APP		
3A	End-Point	ATM Appliance			ATM-A	
3B	APP Service	ATM Driving			APP Service	
4A	APP Service	ATM Driving				APP Service
4C	INFRA	Network Gateway				INFRA GW
E2E	ALL Layers	ATM Channel End-2-End	COD ATM-Apl or ATM APP	ATM-APP ATM-Apl	ATM-Apl APP Service	APP Service INFRA GW

APPENDIX A: HIGH-LEVEL ATM-APP API (ECMAScript 6)

The scope of this appendix is to define the High-Level Javascript (ES6) provider-agnostic API (blue) between the deployer-specific ATM-APP (green) and the Appliance provider-specific software component (orange/grey) like shown in the figure below.

Out of scope are the End-Point Devices Layer SDK of the ATM-APP in green, which will define the deployer-specific UX interface e.g. HTML5 and the ATM Appliance software which will define the low level interface to the Appliance Hardware e.g. XFS in case of MVS ATM Appliance software.

The out of scope are the App Services Layer connections for this appendix, which will only make the context environment in the figure complete. The ATM Appliance software/Hardware in orange has connection with the Appliance Server, the ATM Appliance software has the online provider-agnostic ISO20022 connection with the ATM driving APP Service, while the deployer-specific ATM-APP have a connection with the APP Server.



8 Use Case Framework

The High-Level API will be designed and defined using functional use case examples, to provide more context how the API can be implemented by the ATM-APP and ATM Appliance software.

The API per use case will contain three parts, INPUT – PROCESS – OUTPUT using the responsibilities which are defined in the table below.



UX Interface			
APP Flow			
Transaction Processing			
Hardware Interfaces			
Online ATM driving API			
Security			

The deployer can define his own ATM APP flow using the provider-agnostic API building-blocks defined in this appendix. While the Appliance provider can support these provider-agnostic API building-blocks to support these deployer-specific ATM APPs and flows. The combination of these two are key for a Next-Gen ATM configuration.

8.1 Version 1.0

8.1.1 SERVICE use case

The first use case in IDLE state of the ATM-APP will be the SERVICE use case before the ATM will be in-service (or out-of-service).

Step	ATM-APP	API	ATM Appliance Sw	Notes
1	IDLE, Start-Up Screen Start In-Service →	SERVICE	Validate available services	
2	UX available APPs Based on services	SERVICE	← Provide Available Services	IN-Service Out-of-Service

8.1.2 CARD-AUTHENTICATION use case

Multiple Transactions will use the same authentication use case, based on the magstripe / EMV contact chip card and user PINcode.

Step	ATM-APP	API	ATM Appliance Sw	Notes
1	UX display Read Card →	Authenticate ReadCard		Read Card data
2	UX Display		← Status Read Card	Magstripe, EMV Contact, EMV Contactless
3	UX Display Read PINblock→	Authenticate PINBlock		
	UX Display		SecureKeyboard.startReading () keyPressed Event [1..n] ← Status PINBlock	

8.1.3 INPUT Cash-Out

Step	ATM-APP	API	ATM Appliance Sw	Notes
1	UX Interface Dialogue - Select Amount - Bank Notes			Deployer SDK for UX e.q. HTML5

8.1.4 PROCESS Cash-Out

Step	ATM-APP	API	ATM Appliance Sw	Notes
2	User Interface PROCESS	N.A.		
3	Start Cash-Out Process (amount, banknotes, ...)	ProcessCashOut	WithdrawalTx.authorize (amount, account) - Create Online Message ISO20022 incl. PINBlock - Send Online Message - Receive Online Message	ATM Driver APP Server → Receive Message ← Send Response
4		ProcessCashOut	WithdrawalTx.authorize (amount, account) - Create Online Message ISO20022 incl. PINBlock - Send Online Message - Receive Online Message	
5	User Interface PROCESS Results	N.A.		

8.1.5 OUTPUT Cash-Out

Step	ATM-APP	API	ATM Appliance Sw	Notes
6	User Interface Card Return START CARD-RETURN Process	ReturnCard	Return Card to Cardholder	
7		ReturnCard	← Status Card Returned Event (OK, NOK, ERROR, ...)	
8	START CASH-OUT OUTPUT → (incl.print receipt, options)	CashOut	Validate online Auth Event status OK-Authorized	
9	←User Interface DISPLAY status based on CASH-OUT Results	CashOut	WithdrawalTx.dispense () - dispensingCash Event - cashPresented Event - cashTaken Event ← STATUS CashOut	Fulfil transaction
10	START PRINT-RECEIPT	PrintReceipt	receiptBegin/End Event	
11	CLOSE Transaction	EndSession	<ul style="list-style-type: none"> Commit / Rollback Transactions Send Online Message (on Failure) Write e-journal Remove TransactionData in memory ← Status EndSession	ATM Driver App Server → Receive Message ← Send Response

8.2 Version 2.0 – CASH-IN (Deposit)

8.2.1 INPUT Cash-IN

Step	ATM-APP	API	ATM Appliance Sw	Notes
1	UX Interface Dialogue Deposit to own account			Deployer SDK for UX e.g. HTML5

8.2.2 PROCESS Cash-IN

Step	ATM-APP	API	ATM Appliance Sw	Notes
2	User Interface PROCESS "Insert BankNotes"	DepositCashIN	Count deposit cash-in banknotes	
3		DepositCashIN	Provide cash-in deposit totals currency, numbers, total value per bank note nomination and of total deposit	
4	Confirmation of deposit Cash-IN totals by user			
5	Start Cash-IN Process (deposit own account)	ProcessCashIN	ProcessCashINTx.authorize (DepositCASHIN amount, account) - Create Online Message ISO20022 incl. PINBlock - Send Online Message - Receive Online Message	ATM Driver APP Server → Receive Message ← Send Response
5	User Interface PROCESS Results	N.A.		

8.2.3 OUTPUT Cash-IN

Step	ATM-APP	API	ATM Appliance Sw	Notes
6	User Interface Card Return START CARD-RETURN Process	ReturnCard	Return Card to Cardholder	
7		ReturnCard	← Status Card Returned Event (OK, NOK, ERROR, ...)	
8	START CASH-IN OUTPUT → (incl. print receipt, options)	CashIN	Validate online Auth Event status OK-Authorized	
9	←User Interface DISPLAY status based on CASH-IN Results	CashOut	CASHINTx.deposit () - depositCash Event - cashTaken Event ← STATUS CashIN	Fulfil transaaction
10	START PRINT-RECEIPT	PrintReceipt	receiptBegin/End Event	
11	CLOSE Transaction	EndSession	<ul style="list-style-type: none"> Commit / Rollback Transactions Send Online Message (on Failure) Write e-journal Remove TransactionData in memory ← Status EndSession	ATM Driver App Server → Receive Message ← Send Response

8.3 Version 2.0 - Crypto Currency BUY

Crypto Currency buy use case follows almost the same process flow as CASH-IN Deposit, using the Crypto Currency Wallet Account instead of the Card Account.

8.3.1 INPUT Crypto-BUY

Step	ATM-APP	API	AM Appliance Sw	Notes
1	UX Interface Dialogue Select Crypto Currency			Deployer SDK for UX e.g. HTML5

8.3.2 PROCESS Crypto-BUY

Step	ATM-APP	API	ATM Appliance Sw	Notes
2	User Interface PROCESS "Insert CASH"	DepositCashIN	Count deposit cash-in banknotes	
3		DepositCashIN	Provide cash-in deposit totals currency, numbers, total value per bank note nomination and of total deposit	
4	Confirmation of deposit Cash-IN totals by user			
5	Scan Crypto wallet Account	QRCodeReader	Read Crypto Currency Wallet QRCode	
6	Start Crypto-BUY Process (buy crypto currency)	ProcessCryptoBUY	Process CryptoBUYTX.authorize (DepositCASHIN amount, account) (QRCodeReader crypto currency, wallet accountdata) - Create Online Message ISO20022 - Send Online Message - Receive Online Message	ATM Driver APP Server → Receive Message ← Send Response
7	User Interface PROCESS Results	N.A.		

8.3.3 OUTPUT CRYPTO-BUY

Step	ATM-APP	API	ATM Appliance Sw	Notes
8	START CRYPTO-BUY CASH-IN OUTPUT → (incl. print receipt, options)	CashIN	Validate online Auth Event status OK-Authorized	
9	←User Interface DISPLAY status based on CASH-IN Results	CashOut	CASHINTx.deposit () - depositCash Event - cashTaken Event ← STATUS CashIN	Fulfill transaction
10	START PRINT-RECEIPT	PrintReceipt	receiptBegin/End Event	Incl. Crypto Currency Amount
11	CLOSE Transaction	EndSession	• Commit / Rollback Transactions • Send Online Message (on Failure) • Write e-journal Remove TransactionData in memory ← Status EndSession	ATM Driver App Server → Receive Message → Send Crypto Amount to Crypto Wallet ← Send Response

8.4 Version 2.0 - Crypto Currency SELL

Crypto Currency sell use case follows almost the same process flow as CASH-OUT Withdrawal, loading the Crypto Currency Wallet Account instead of providing CASH BankNotes and Coins.

8.4.1 INPUT CRYPTO-SELL

Step	ATM-APP	API	ATM Appliance Sw	Notes
1	UX Interface Dialogue - Select Crypto Currency - Select Crypto Amount to Sell - Enter Mobile phone number			Deployer SDK for UX e.g. HTML5
2	Request ATM Crypto QRCode →	ATMCryptoTrxAccount		
3	Display ATM Crypto QRCode		Send QRCode data of ATM Crypto Account	
3				<u>COD-of customer</u> Crypto Currency Wallet READ QRCode from ATM.
4				<u>Crypto Currency Wallet</u> <u>SEND Crypto Currency Amount</u> <u>to ATM Crypto Account</u>

8.4.2 PROCESS CRYPTO-SELL

Step	ATM-APP	API	ATM Appliance Sw	Notes
5	User Interface PROCESS	N.A.		
6	Start Crypto-SELL Process (amount, crypto currency, ...)	ProcessCryptoSELLt	CryptoSellTx.authorize (amount, crypto currency, user mobile phone number) - Create Online Message ISO20022 - Send Online Message - Receive Online Message	ATM Driver APP Server → Receive Message Validate Crypto Currency Amount is received at ATM Crypto Account. SMS Crypto Sell status and One Time Password (OTP) to mobile phone number. ← Send Response (including encrypted OTP)
7	User Interface PROCESS Results	N.A.		

8.4.3 OUTPUT CRYPTO-SELL

Step	ATM-APP	API	ATM Appliance Sw	Notes
8	START CRYPTO SELL OUTPUT → (incl.print receipt, options) Enter received One Time Password (OTP) on mobile phone	CashOut	Validate online Auth Event status OK-Authorized/Received	Including Validate User entered One Time Password with received ATM Driver APP Server OTP
9	←User Interface DISPLAY status based on CASH-OUT Results	CashOut	WithdrawalTx.dispense () - dispensingCash Event - cashPresented Event - cashTaken Event ← STATUS CashOut	Fulfil transaction
10	START PRINT-RECEIPT	PrintReceipt	receiptBegin/End Event	
11	CLOSE Transaction	EndSession	• Commit / Rollback Transactions • Send Online Message (on Failure) • Write e-journal Remove TransactionData in memory ← Status EndSession	ATM Driver App Server → Receive Message ← Send Response

9 JavaScript SPECIFICATIONS (2019, version 1.0)

The NextGen ATMIA is implemented as a set of Javascript Objects that implement Service APIs. By decomposing the API across a suite of Javascript Objects, the API is focused, simplified and extensible. The NextGen object is the main entry point for applications and provides access to the following Services.

Version 1.0 Services;

- Session (9.2)
- CardReader (9.3)
- Card (9.4)
- PIN (9.5)
- QRCodeReader (9.6)
- Withdrawal (9.7)
- Receipt (9.8)
- Error Handling (9.9)

9.1 NextGen Object

The API exposes the set of Javascript Objects, available from the Registry, that implement the NextGen Javascript API.

The NextGen Instance exposes the following functions.

- `getService(serviceName) : object`

9.1.1 Get Service

`NextGen.getService(serviceName): object`

Returns the corresponding Javascript Object that implements the API.

9.1.1.1 Function arguments

This function call requires the following arguments.

Argument	Type	Required	Description
serviceName	string	Required	The Service Name requested, either "Session", "Card", "QRCode", "PIN", "Withdrawal", "Receipt"

9.1.1.2 Return value

This function returns the following values.

Type	Required	Description
object	Required	A Javascript Object implementing the Service API

9.2 Session Service API

The Session Service API is exposed to allow the NextGen application to start an interaction with a consumer. Applications retrieve this object on start up from the NextGen Registry (“Session”).

The Session Service instance is required to expose the following functions.

- `getInfo(): SessionInfo`
- `startInService (inServiceListener): void`
- `startSession (): void`
- `closeSession (): void`

The coming sections provide a detailed description of these functions, and the objects and flows that are involved.

9.2.1 Get Session Information

`SessionService.getInfo(): SessionInfo`

Returns information about the ATM including whether it is in service and the available card readers, etc. See the *SessionInfo* object for a detailed description of the return values.

9.2.1.1 Function arguments

This call does not require parameters.

9.2.1.2 Return value

The object returned will be an instance of *SessionInfo* containing the following members.

Field	Type	Required	Description
<code>status</code>	boolean	Required	Describes whether the Session is ready for interaction.
<code>readers</code>	string[]	Required	A list containing the readers that are available for this ATM. Allowing the Session consumer experience to correctly display instructions to the consumer.
<code>readerId</code>	string	Required	The identifier of the available reader. <ul style="list-style-type: none"> • “QRCode” • “contactless” • “dip” • “swipe” • “latchdip” • “motorized”

9.2.2 Start In Service

```
SessionService.startInService(inServiceListener): void
```

Applications should call this function during/after their start-up sequence and after each customer session. Calling this function, will inform the API that the application is ready to provide service. The application will receive notifications that interaction was initiated via the *inServiceListener* it provided.

9.2.2.1 Function arguments

This function requires that an instance of *inServiceListener* is provided.

Argument	Type	Required	Description
<i>inServiceListener</i>	Object	Required	Object containing callback functions for sessions initiated by the ATM.
<code>ready ()</code>	Function	Required	This function will be invoked when the ATM is in-service and waiting for a Consumer to interact with the device. The ATM should display the “attract sequence” until a consumer is present.
<code>notReady (error)</code>	Function	Required	This function will be invoked when the ATM is out-of-service and cannot interact with a Consumer. The ATM should display the “out-of-service” experience until the error is corrected. Detailed information is provided through the <i>error</i> object.
<code>cardDetected ()</code>	function	Required	Indicates a card was presented to one of the ATM’s readers.
<code>cardAccepted ()</code>	function	Required	Indicates the card was read successfully.
<code>failedCardAccept ()</code>	Function	Required	Indicates a failure occurred accepting the card
<code>sessionStarted(cardProvided)</code>	Function	Required	This function will be invoked when the ATM is initiating a session. E.g. when it registers a card is being presented through one of its readers or via the <code>StartSession</code> function. The function receives one argument. - <code>cardProvided: boolean</code> Indicates whether a card was already provided and successfully read.
<code>sessionEnded ()</code>	Function	Required	This function will be involved after <code>endSession</code> and indicates that session with the consumer is complete. To start a session with the next consumer – the Application must call <code>startInService</code>

9.2.2.2 Return value

This function does not return anything. Instead feedback is provided through the functions defined in the *sessionListener* argument.

9.2.2.3 Exceptions

This function will throw an exception if one or more of the required arguments were not provided.

9.2.3 Start Session

```
SessionService.startSession(): void
```

Applications can optionally initiate a session for a cardholder (for example, by touching a screen or via their mobile device). This function can be called to start such a session. Note, if a session is started using a Reader – this function does not need to be called.

9.2.3.1 Function arguments

This function does not require any arguments.

9.2.3.2 Return value

This function does not return anything. Instead feedback is provided through the functions defined in the *inServiceListener* argument provided in the startInService call.

9.2.3.3 Exceptions

This function will throw an exception if a session is already active.

9.2.4 End Session

```
SessionService.endSession(): void
```

Applications must end a session at the end of the interaction with the cardholder. Once the session is ended the Application must call **startInService** to start an interaction with the next consumer.

9.2.4.1 Function arguments

This function does not require any arguments.

9.2.4.2 Return value

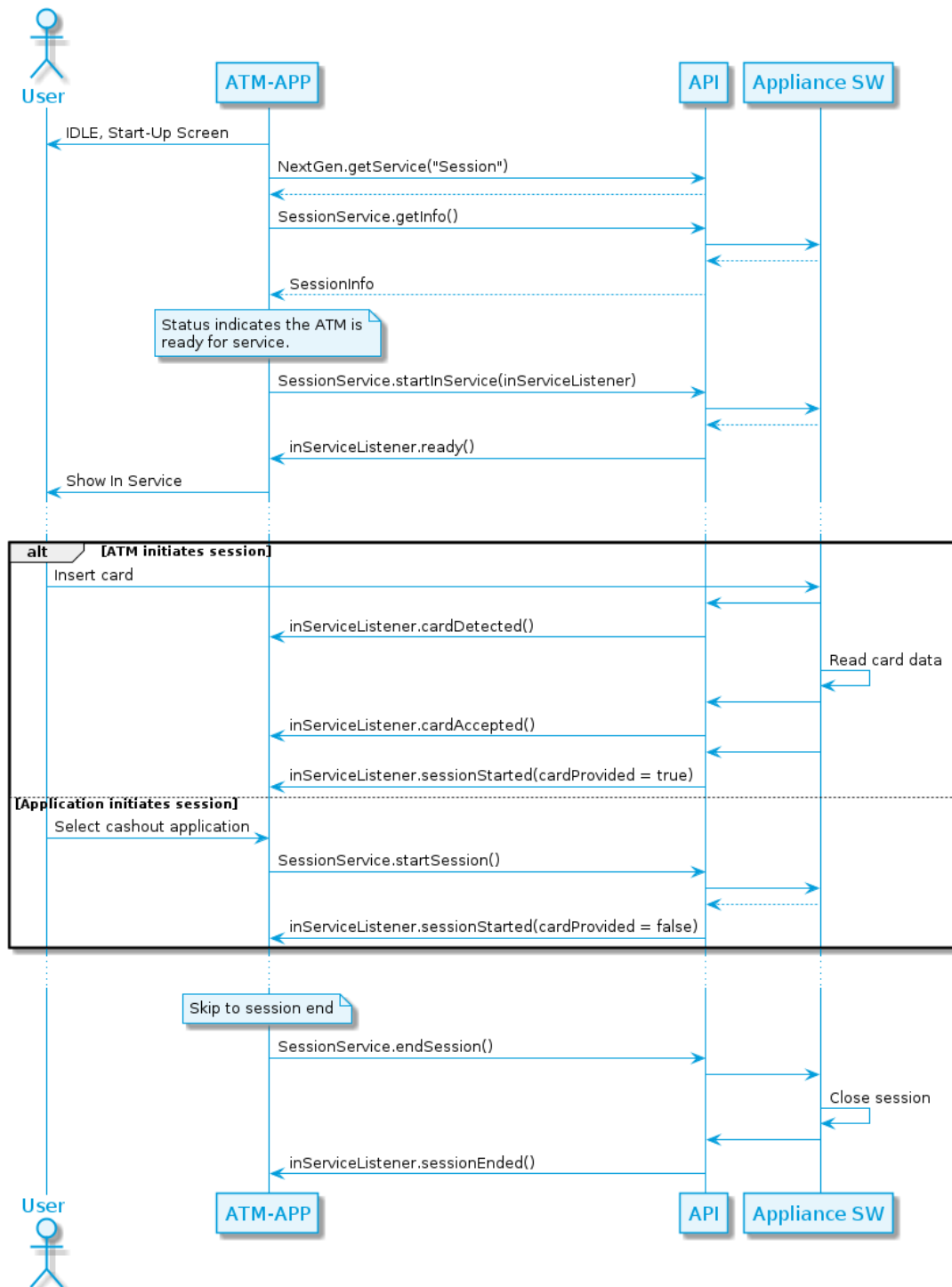
This function does not return anything. Instead feedback is provided through the functions defined in the *inServiceListener* argument provided in the startInService call.

9.2.4.3 Exceptions

This function will throw an exception if a session is not active.

9.2.5 Flows

IN SERVICE flow



9.3 CardReader Service API

The CardReader Service API is exposed to allow the NextGen application to start an interaction with a card reader. Applications retrieve this object on start up from the NextGen Registry ("CardReader").

The Card Service instance is required to expose the following functions.

- `acceptCard (readCardListener): void`
- `ejectCard (returnCardListener): void`

The coming sections provide a detailed description of these functions, and the objects and flows that are involved.

9.3.1 Accept Card

`CardReaderService.acceptCard(readCardListener): void`

Requests the ATM to listen to its card readers provide events when a card was provided. This method is intended to be used when `SessionService.sessionStarted(cardProvided)` was invoked with `cardProvided = false`, or when the current session was initiated by the application. This API provides limited card data to the application.

9.3.1.1 Function arguments

This function requires the following arguments.

Argument	Type	Required	Description
<code>acceptCardListener</code>	object	Required	Object containing callback functions that will be used by the API to provide asynchronous events containing information whether card data was read.
<code>cardDetected ()</code>	function	Required	Indicates a card was presented to one of the ATM's readers.
<code>cardAccepted ()</code>	function	Required	Indicates the card was read successfully.
<code>failedCardAccept()</code>	function	Required	Indicates a failure occurred accepting the card
<code>error(error)</code>	function	Required	This function will be called when the ATM was unable to obtain card data. Error details are provided through the <code>error</code> object.

9.3.1.2 Return value

This function does not return anything. Instead feedback is provided through the functions defined in the `readCardListener` argument.

9.3.1.3 Exceptions

This function will throw an exception if one or more of the required arguments were not provided.

9.3.2 Eject Card

```
CardReaderService.ejectCard(ejectCardListener): void
```

Instructs the ATM to eject the card. Event updates will be provided through the callbacks that are provided in the `ejectCardListener` object.

9.3.2.1 Function arguments

This function requires the following arguments.

Argument	Type	Required	Description
<code>ejectCardListener</code>	object	Required	Object containing callback functions that will be used by the API to provide asynchronous updates during the card eject process.
<code>cardPresented()</code>	function	Required	Indicates the card was presented to the consumer.
<code>failedCardPresent()</code>	function	Required	Indicates that a failure occurred in ejecting the card
<code>cardTaken()</code>	function	Required	Indicates the card was taken.
<code>cardCaptured()</code>	function	Required	Indicates that card was not taken and subsequently captured.
<code>failedCardCapture()</code>	function	Required	Indicates that a failure occurred attempting to capture a card
<code>error(error)</code>	function	Required	This function will be called when the ATM was unable to return the card. E.g. when the contactless reader was used to read the card data. Error details are provided through the <code>error</code> object.

9.3.2.2 Return value

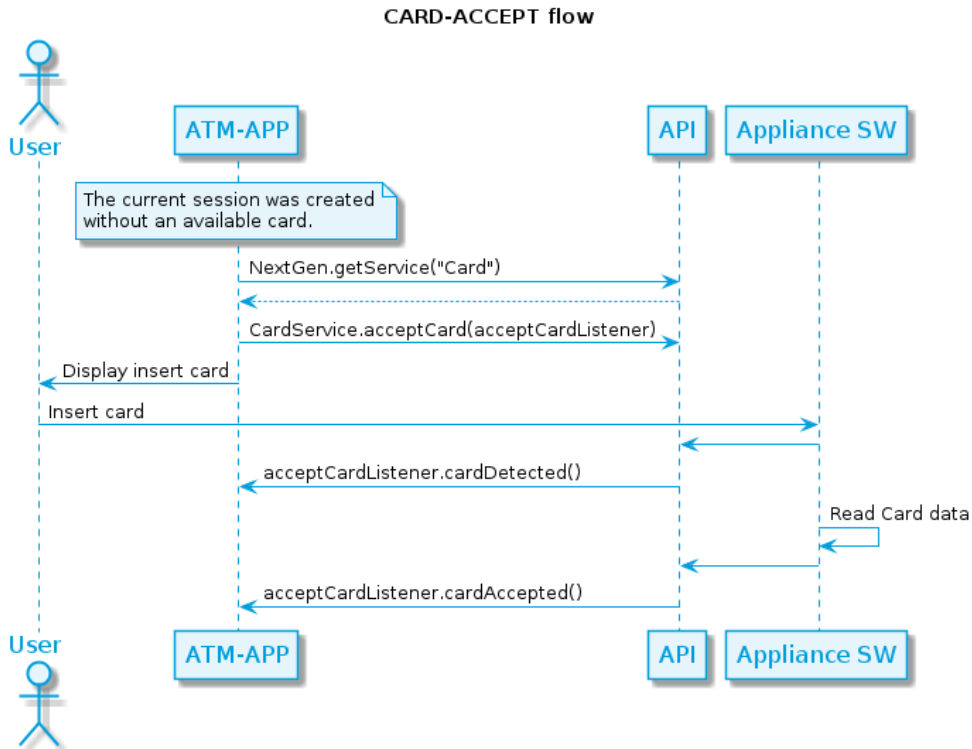
This function does not return anything. Instead feedback is provided through the functions defined in the `ejectCardListener` argument.

9.3.2.3 Exceptions

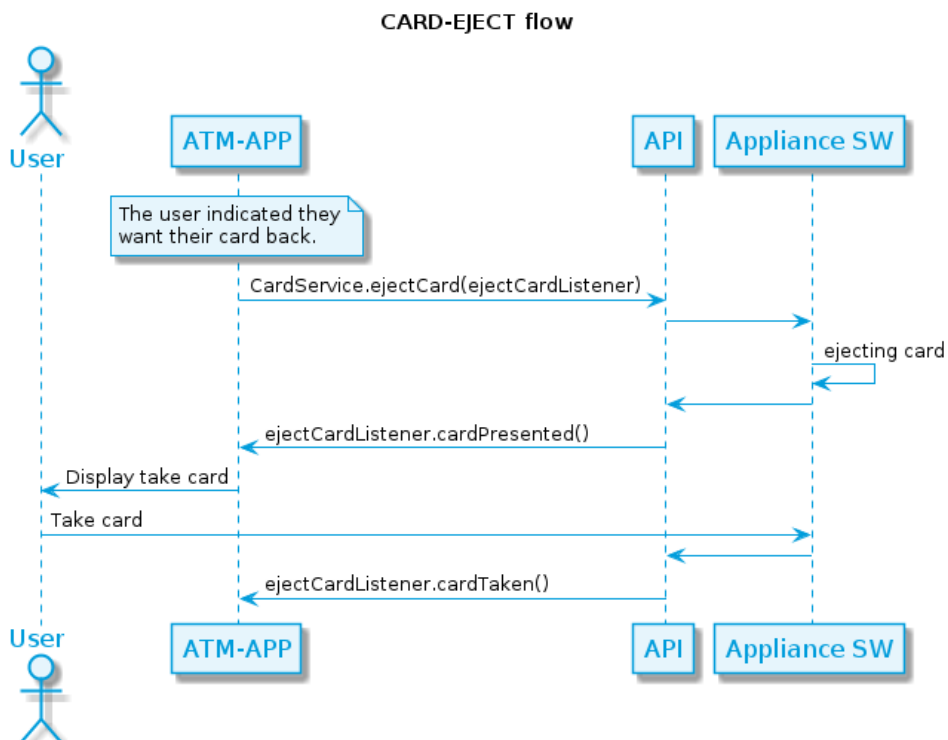
This function will throw an exception if one or more of the required arguments were not provided.

9.3.3 Flows

9.3.3.1 Card Accept Flow



9.3.3.2 Card Eject Flow



9.4 Card Service API

The Card Service API is exposed to allow the NextGen application to start an interaction with a card. Applications retrieve this object on start up from the NextGen Registry (“Card”).

The Card Service instance is required to expose the following functions.

- `getInfo(): CardInfo`
- `selectApplication (applicationId, applicationSelectionListener): void`

The coming sections provide a detailed description of these functions, and the objects and flows that are involved.

9.4.1 Get Info

`CardService.getInfo(): CardInfo`

Requests the details of the Card from the ATM.

9.4.1.1 Function arguments

This call does not require parameters.

9.4.1.2 Return value

The object returned will be an instance of *CardInfo* containing the following members.

Field	Type	Required	Description
<code>status</code>	boolean	Required	Describes whether the Card Service is ready for interaction.
<code>authorizationMethod</code>	string	Required	Describes the method used with this card for transaction authorization. Must be one of the following values: “ActionRequired”, “MagneticStrip”, “MagneticStripFallback”, “EMVApplication”, “NoAuthorization”
<code>actionRequired</code>	string	Required	Describes the next action to be taken before the card can be used for authorization: “ReadMagneticStripe”, “ReadChip”, “SelectEMVApplication”, “UseSingleCard”, “SeeDeviceForInstructions”, “None”
<code>minimumPinLength</code>	number	Required	The minimum length of PIN (in number of digits) that is associated with this card
<code>maximumPinLength</code>	number	Required	The maximum length of PIN (in number of digits) that is associated with this card.
<code>preferredLanguages</code>	string[]	Required	Array of string representing ISO 639-1 language codes
<code>selectedApplication</code>	object	Optional	Currently selected EMV Application or null
<code>availableApplications</code>	list of emvApplication objects	Optional	
<code>applicationId</code>	string	Required	An identifier that uniquely identifies an application on an EMV-enabled card
<code>label</code>	string	Required	A description of the application using only characters from the EMV Common Character Set

preferredName	string	Required	A description of the application in the cardholders' own language using characters from CodeTable
preferredNameCodeTable	number	Required	The ISO/IEC 8859 code table to be used to display the preferredName. An integer > 0.
confirmationRequired	boolean	Required	True if the consumer must confirm this application selection, otherwise false.

9.4.2 Select Application

```
CardService.selectApplication(applicationId,applicationSelectionListener):void
```

Requests the details of the Card from the ATM.

9.4.2.1 Function arguments

This function requires the following arguments.

Argument	Type	Required	Description
applicationId	String	Required	The identifier that uniquely identifies an application on an EMV-enabled card
applicationSelectionListener	object	Required	Object containing callback functions that will be used by the API to provide asynchronous events containing information on application selection.
applicationSelected()	function	Required	This function is called when the requested EMV application has been successfully selected
applicationSelectionFailed()	function	Required	This function is called when the EMV application could not be selected.

9.4.2.2 Return value

This function does not return anything. Instead feedback is provided through the functions defined in the *applicationSelectionListener* argument.

9.4.2.3 Exceptions

This function will throw an exception if one or more of the required arguments were not provided.

9.5 PIN Service API

The PIN Service API is exposed to allow the NextGen application to initiate secure PIN entry. Applications retrieve this object on start up from the NextGen Registry (“PIN”).

The PIN Service instance is required to expose the following functions.

- `collectPIN (collectPINListener): void`

The coming sections provide a detailed description of these functions, and the objects and flows that are involved.

9.5.1 Collect PIN

`PINService.collectPIN(collectPINOptions, collectPINListener): void`

Requests the ATM to obtain the PIN code from the end-user. This function requires that a card has been successfully read within the current session. In limited PIN data collection is exposed to the application. PIN encryption will occur automatically at the end of the Collect PIN function.

9.5.1.1 Function arguments

This function requires the following arguments.

Argument	Type	Required	Description
<code>options</code>	object	Required	Object containing the collect PIN options
<code>minLength</code>	number	Optional	Minimum number of digits in the PIN. If defined must be an integer greater than 0
<code>maxLength</code>	number	Optional	Maximum number of digits in the PIN. If defined must be an integer greater than 0
<code>autoEnd</code>	boolean	Optional	Defines if PIN collection ends automatically when the length of the PIN = <code>maxLength</code>
<code>keys</code>	string[]	Optional	Array of strings that define which keys are to be enabled. <ul style="list-style-type: none"> • “enter” – terminates input • “cancel” – terminates input • “clear” – non terminating • “backspace” – non terminating • “digit” – non terminating • “00” – non terminating • “000” – non terminating • “help” – non terminating • “dec” – non terminating
<code>collectPINListener</code>	object	Required	Object containing callback functions that will be used by the ATM to provide the application with feedback during the validate PIN process.
<code>keyPressed(key, terminated)</code>	function	Required	This function will be invoked when a key press occurs during PIN entry: <ul style="list-style-type: none"> - Key – String as defined in keys - Terminated – Boolean Indication that entry is complete
<code>error(error)</code>	function	Required	This function will be called when the ATM was unable to obtain the PIN. Error details are provided through the <code>error</code> object.

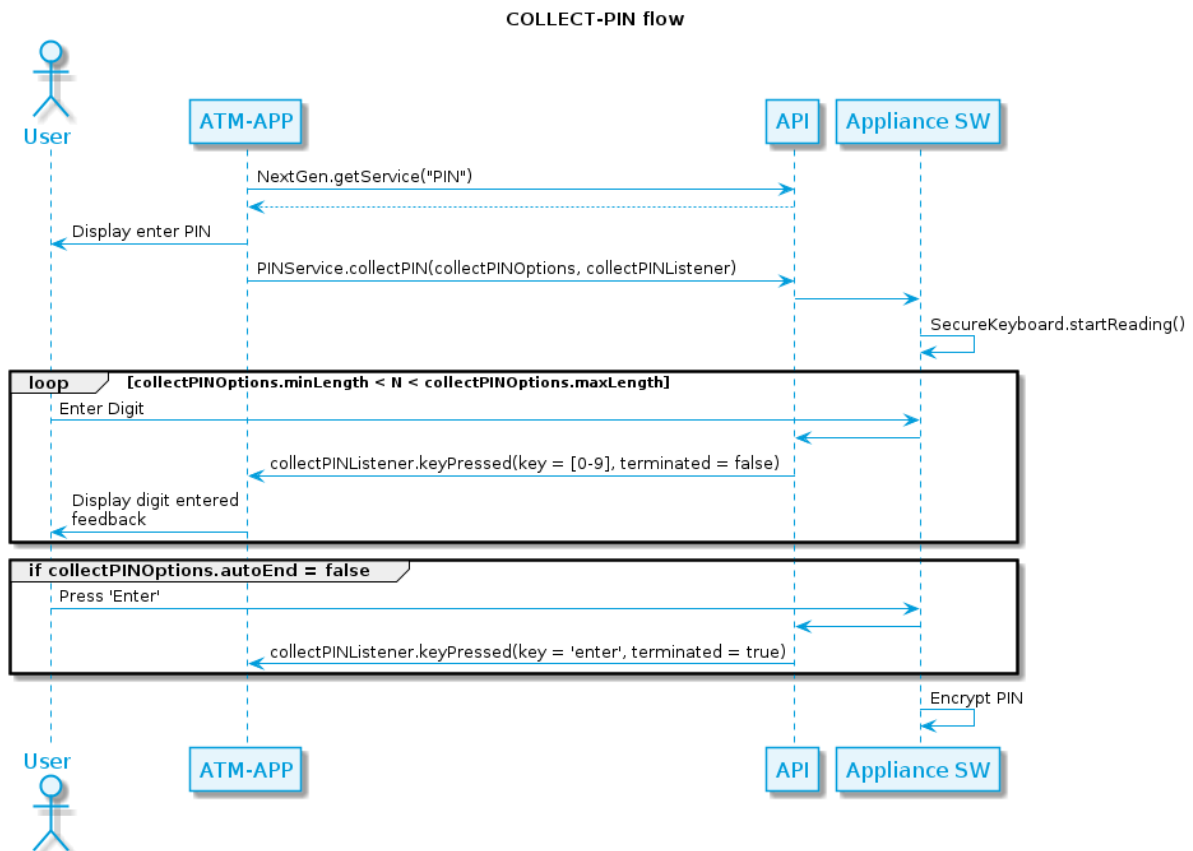
9.5.1.2 Return value

This function does not return anything. Instead feedback is provided through the functions defined in the *collectPINListener* argument.

9.5.1.3 Exceptions

This function will throw an exception if one or more of the required arguments were not provided.

9.5.2 Flow



9.6 QRCode Reader Service API

The QRCode Reader Service API is exposed to allow the NextGen application to instruct the ATM to read a QRCode. Applications retrieve this object on start up from the NextGen Registry (“QRCode”).

The QRCode Reader Service instance is required to expose the following functions.

- `getData(): QRCodeData`

The coming section provides a detailed description of this function, and the objects and flows that are involved.

9.6.1 Get Info

```
QRCodeReaderService.getData(qrCodeReaderListener): void
```

Requests the ATM to attempt to read a QRCode via the QRCode reader.

9.6.1.1 Function arguments

This requires the following parameters:

Field	Type	Required	Description
<code>qrCodeReaderListener</code>	object	Required	Object containing callback functions that will be used by the ATM to provide the application with feedback about reading the QRCode.
<code>readQRCodeSuccessful(data)</code>	function	Required	This function will be invoked when reading the QRCode was successful. <ul style="list-style-type: none"> - <code>data</code> – String containing Base64 encoded QRCode data. This string will contain the authorization token required to finish the payment.
<code>error(error)</code>	function	Required	This function will be called when the ATM was unable to read the QRCode. Error details are provided through the <code>error</code> object.

9.6.1.2 Return value

This function does not return anything. Instead feedback is provided through the functions defined in the `qrCodeReaderListener` argument.

9.6.1.3 Exceptions

This function will throw an exception if one or more of the required arguments were not provided.

9.7 Withdrawal Service API

The Withdrawal Service API is exposed to allow the NextGen application to initiate a cash withdrawal transaction with fulfilment. Applications retrieve this object on start up from the NextGen Registry (“Withdrawal”).

The Withdrawal Service instance is required to expose the following functions.

- `getInfo ()`: `WithdrawalInfo`
- `authorize (authorizeHandler)`: `void`
- `dispense (dispenseHandler)`: `void`

The coming sections provide a detailed description of these functions, and the objects and flows that are involved.

9.7.1 Get Withdrawal Information

`WithdrawalService.getInfo(): WithdrawalInfo`

Returns information about the Withdrawal information for the ATM.

9.7.1.1 Function arguments

This call does not require parameters.

9.7.1.2 Return value

The object returned will be an instance of *WithdrawalInfo* containing the following members.

Field	Type	Required	Description
<code>status</code>	boolean	Required	Describes whether the Withdrawal Service is ready for interaction.
<code>denominations</code>	list of amount objects	Required	A list containing the denominations available from the ATM.
<code>currency</code>	string	Required	The currency of the amount. ISO-4217
<code>amount</code>	number	Required	The value of the amount.

9.7.2 Authorize

`WithdrawalService.authorize (authorizeRequest, authorizeListener): void`

Instructs the ATM to start processing a Withdrawal with the provided values. The ATM will process the transaction on the Network of the card that was read. Event updates will be provided through the callbacks that are provided in the *authorizeListener* object.

9.7.2.1 Function arguments

This function call requires the following arguments.

Argument	Type	Required	Description
<code>authorizeRequest</code>	object	Required	Object containing the withdrawal request
<code>amount</code> - <code>currency</code> - <code>value</code>	amount object string number	Required	Amount (object) to authorize
<code>account</code> - <code>accountNumber</code> - <code>bankId</code> - <code>otherId</code>	account object string string string	Optional	Account (object) to authorize
<code>token</code>	string	Optional	AuthToken data to be send to the acquirer. Token may be obtained in a cardless transaction, e.g. via QRCode.
<code>notemix</code> - <code>count</code> - <code>denomination</code> <code>currency</code> <code>value</code>	denomination[] number amount object	Optional	Notemix of cash to withdraw.
<code>ejectCard</code>	String	Optional	"CardBeforeCash", "CardAfterCash", "DoNotEject"
<code>authorizeListener</code>	object	Required	Object containing callback functions that will be used by the ATM to provide the application with feedback during the cash out process.
<code>authorized(</code> <code>event</code> <code>)</code>	function	Required	This function will be called when the host system has authorized the withdrawal.
<code>unableToAuthorize(</code> <code>responseCode</code> <code>)</code>	function	Required	This function will be called when the host system has not authorized the withdrawal. Response code indicates the cause of the failure.
<code>error (</code> <code>error</code> <code>)</code>			This function will be called when an error occurred while processing the Withdrawal. Error details are provided through the <i>error</i> object.

9.7.2.2 Return value

This function does not return anything. Instead feedback is provided through the functions defined in the *authorizeListener* argument.

9.7.2.3 Exceptions

This function will throw an exception if one or more of the required arguments were not provided. This function will throw an exception if there is no card-data available within the active session.

9.7.1 Dispense

```
WithdrawalService.dispense( dispenseListener): void
```

Instructs the ATM to fulfil the Withdrawal that was processed via the *WithdrawalService.authorize* function. Event updates will be provided through the callbacks that are provided in the *dispenseListener* object.

9.7.1.1 Function Arguments

This function requires the following arguments.

Argument	Type	Required	Description
<code>dispenseListener</code>	object	Required	Object containing callback functions that will be used by the API to provide asynchronous updates during the output cash process.
<code>cashPresented ()</code>	function	Required	Indicates cash has been presented to the consumer
<code>cashTaken ()</code>	function	Required	Indicates cash has been taken by the consumer
<code>cashRetracted ()</code>	function	Required	Indicates cash was retracted after being presented
<code>failedRetractCash ()</code>	function	Required	Indicated a failure retracting cash
<code>cardEjected()</code>	function	Required	Indicates the card was ejected.
<code>failedEjectCard()</code>	function	Required	Indicates that a failure occurred in ejecting the card
<code>cardTaken()</code>	function	Required	Indicates the card was taken.
<code>cardCaptured()</code>	function	Required	Indicated that card was not taken and subsequently captured.
<code>failedCaptureCard()</code>	function	Required	Indicated that a failure occurred attempting to capture a card
<code>error(error)</code>	function	Required	This function will be called when the ATM was unable to return the card. E.g. when the contactless reader was used to read the card data. Error details are provided through the <i>error</i> object.

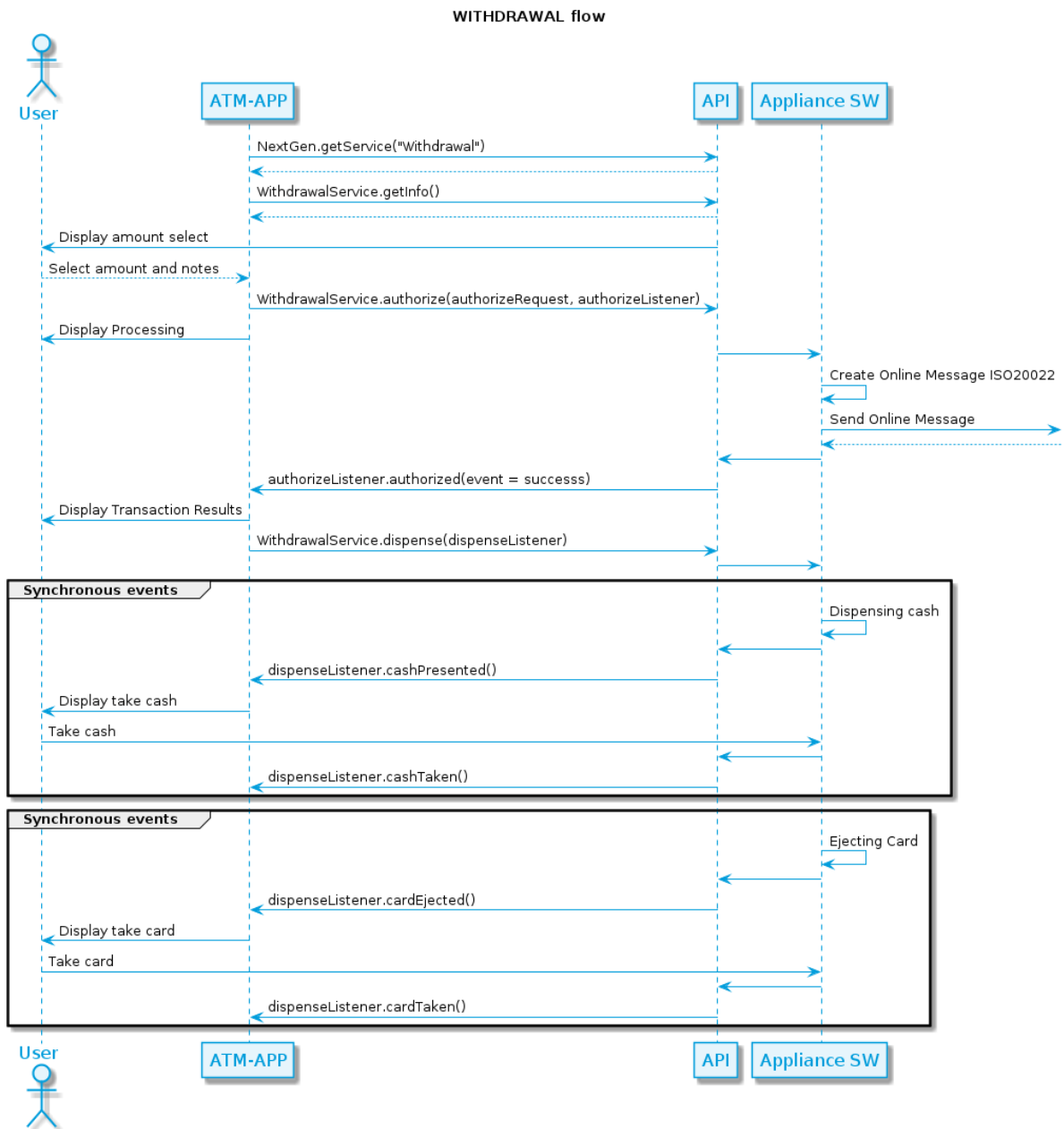
9.7.1.2 Return value

This function does not return anything. Instead feedback is provided through the functions defined in the *dispenseListener* argument.

9.7.1.3 Exceptions

This function will throw an exception if one or more of the required arguments were not provided. This function will throw an exception if no transaction was processed previously within the session. (*WithdrawalService.authorize(...)* was not completed successfully.)

9.7.2 Flow



9.8 Receipt Service

The Receipt Service API is exposed to allow the NextGen application to initiate a receipt print. Applications retrieve this object on start up from the NextGen Registry (“Receipt”).

The Withdrawal Service instance is required to expose the following functions.

- `print (additionalData, printListener): void`

9.8.1 Print Receipt

`Receipt.print(additionalData, printReceiptListener): void`

Instructs the ATM to print a receipt. The information on the receipt depends on what processes were executed. Extra data might be added to the receipt by providing *additionalData*.

9.8.1.1 Function arguments

This function requires the following arguments to be provided.

Argument	Type	Required	Description
<code>additionalData</code>	string	Optional	Additional text or information that will be printed at the end of the receipt.
<code>printListener</code>	object	Required	Object containing callback functions that will be used by the API to provide asynchronous updates during the print receipt process.
<code>pageEnd ()</code>	function	Required	Indicated the end of a page -
<code>documentEnd ()</code>	function	Required	
<code>error(error)</code>	function	Required	This function will be called when the ATM was unable to print the receipt. E.g. when the printer paper has run out. Error details are provided through the <i>error</i> object.

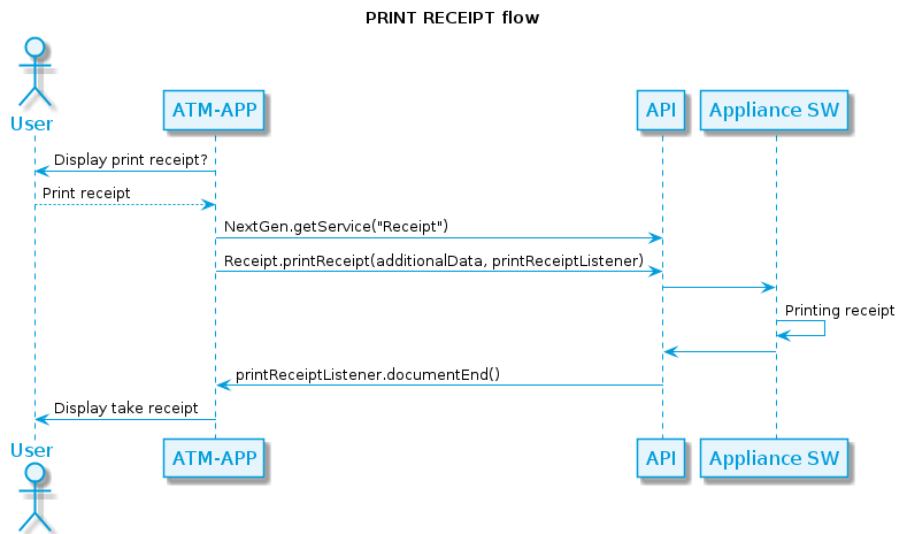
9.8.1.2 Return value

This function does not return anything. Instead feedback is provided through the functions defined in the *returnCardListener* argument.

9.8.1.3 Exceptions

This function will throw an exception if one or more of the required arguments were not provided. This function will throw an exception if the *additionalData* was too long.

9.8.2 Flow



9.8.3 Error Format

Whenever an error occurs an error object will be returned through the listeners error method. All error objects have the following format.

Field	Type	Required	Description
errorId	string	Required	This ID is generated and unique per function call.
errors	list	Required	A list of error objects, one for every error encountered. At least one must be provided.
errorCode	string	Required	The Code of the error that has occurred. This code can be used to find more information about the error.
message	string	Required	A descriptive message of the error.
property	string	Optional	The property for which the error occurred.

10 JavaScript SPECIFICATIONS (2020, version 2.0)

In addition to the previous chapter 9, JavaScript Specifications 2019, version 1.0, the following additional Services are available in the 2020 version.

Version 2.0 Additional Services;

- MixedMedia (10.2)
- Deposit (10.3)
- Crypto-Buy (10.4)
- Crypto-Sell (10.5)

10.1 NextGen Object

The *serviceName* argument of the *NextGen.getService* function (see version 1.0, paragraph 9.1) can additionally be “Deposit”, “MixedMedia”, “cryptoBuy” and “cryptoSell”.

10.2 Mixed Media Service API

The Mixed Media API is exposed to allow the NextGen application to collect the deposit of mixed media items (cash and/or coins) that can be used with either a Deposit, Payment or other Service that requires media item capture. Applications retrieve this object on start up from the NextGen Registry (“MixedMedia”). The Mixed Media Service is a *Collecting Device Service*.

The Mixed Media Service instance is required to expose the following functions.

- getInfo (): MixedMediaInfo
- start (mixedMediaHandler): void
- cancel (): void
- collectMedia (): void
- stopCollecting (): void
- retainMedia (): void
- captureMedia (): void

The coming sections provide a detailed description of these functions, and the objects and flows that are involved.

10.2.1 Get Mixed Media Information

MixedMediaService.getInfo(): MixedMediaInfo

Returns information about the MixedMedia collection capabilities for the ATM.

10.2.1.1 Function arguments

This call does not require parameters.

10.2.1.2 Return value

The object returned will be an instance of *MixedMediaInfo* containing the following members.

Field	Type	Required	Description
status	boolean	Required	Describes whether the Mixed Media Service is ready for interaction.
mediaTypesAccepted	mediaTypes[]	Required	A list containing the mixed media types that can be accepted by the ATM. Each member of the array must be an object as described below.
identifier	string	Required	The identifier of the mixed media collector. Should be used when providing mixedMediaConfiguration.
type	string	Required	Type of the accepted Mixed Media. Must be one of the following values: “bankNotes”, “coins”.
acceptedDenominations - currency - value	denomination[] string number	Optional	A list of Denominations, required when the type of the accepted media type is “bankNotes” or “coins”. The list indicates what coins or banknotes are accepted.

10.2.2 Start

`MixedMediaService.start (mixedMediaConfiguration, mixedMediaListener): void`

Instructs the ATM to initialize the processing a media acceptor. Event updates will be provided through the callbacks that are provided in the *mixedMediaListener* object for all operations of the Mixed Media service.

10.2.2.1 Function arguments

This function call requires the following arguments.

Argument	Type	Required	Description
<code>mixedMediaConfiguration</code>	object	Required	Object containing the mixed media configuration
<code>mixedMediaIdentifier</code>	string	Required	The identifier of the mixed media that should start collecting.
<code>mixedMediaListener</code>	object	Required	Object containing callback functions that will be used by the ATM to provide the application with feedback during the mixed media process.
<code>startComplete()</code>	function	Required	This function will be called when the device has started with the given configuration.
<code>readyToCollect()</code>	function	Required	This function will be called when the ATM is ready to start collecting media. This event is required as there can be a significant delay between issuing the <code>collectMedia()</code> function and the device being ready to accept media (e.g. the time required to open a shutter etc.)..
<code>mediaInserted()</code>	function	Required	This function will be called when the terminal is able to detect that something has been inserted into the device.
<code>mediaDetected (mediaDetected - identifier - type - quantity - denomination - currency - amount)</code>	function MediaItem object string number Amount object string number	Required	This function will be called when the ATM is able to recognize a collected bank note or coin. The function parameter will an object describing the detected media and are described by BankNote or Coin. <i>Note that reporting the detection of bank notes and coins is optional. It is dependent on the capabilities of the hardware.</i>
<code>mediaCollected (mediaCollected)</code>	function MediaItems[]	Required	This function indicates the end of the <code>collectMedia()</code> operation and is called when the terminal has successfully collected and processed a media bundle from the consumer. This event will only be reported when: <ol style="list-style-type: none"> the flap or slot of the device is closed the internal device operation and the recognition processing (for each of the media inserted) is finished The function parameter will an object summarizing the detected media. MediaItem has the structure below. Type is required, whereas notemix, coinmix must be provided based on the type. Types can be "bankNotes", "coins", <pre>{ type: string notemix: object[] coinmix: object[] }</pre>
<code>noMediaCollected (reason)</code>	function string	Required	This function indicates the end of the <code>collectMedia()</code> operation and will be called when no new media was collected during that operation. The function parameter will be a string describing the reason why no media was collected and will be one of the following values: <ul style="list-style-type: none"> <code>noValidMedia</code> - the consumer inserted media but all media has been returned as invalid and taken by the consumer.

				<ul style="list-style-type: none"> • timeout - the consumer failed to insert any media before the flap or slot was closed. • cancelled - the consumer cancelled the collection without inserting any media. • deviceFailure - media could not be accepted as there was a hardware failure that prevented media from being inserted into the device.
<code>mediaRetained ()</code>	function	Required		This function indicates the end of the <code>retainMedia()</code> operation and will be called when the terminal has successfully moved the media item(s) to the safe.
<code>mediaRetainedFailed (reason)</code>	function string	Required		<p>This function indicates the end of the <code>retainMedia()</code> operation and will be called when the terminal fails to move the media item(s) to the safe. The function parameter will be a string describing the failure and will be one of the following values:</p> <ul style="list-style-type: none"> • noMedia - there is no media present in the escrow. • returnsNotTaken - the consumer failed to remove the media returned. • jammed - the device jammed retaining the media. The media remains in the device.
<code>mediaCaptured ()</code>	function	Required		This function indicates the end of the <code>captureMedia()</code> operation and will be called when the terminal has successfully moved the media item(s) to the retract location.
<code>mediaCapturedFailed (reason)</code>	function string	Required		<p>This function indicates the end of the <code>captureMedia()</code> operation and will be called when the terminal fails to move the media item(s) to the retract location. The function parameter will be a string describing the failure and will be one of the following values:</p> <ul style="list-style-type: none"> • noMedia - there is no media present in the escrow. • jammed - the device jammed capturing the media. The media remains either at the mouth, or stuck within, the device.
<code>error (error)</code>				This function will be called when an error occurred while processing the Withdrawal. Error details are provided through the <code>error</code> object.

10.2.2.2 Return value

This function does not return anything. Instead feedback is provided through the functions defined in the `mixedMediaListener` argument.

10.2.2.3 Exceptions

This function will throw an exception if one or more of the required arguments were not provided.

10.2.3 Cancel

```
MixedMediaService.cancel(): void
```

Instructs the ATM to cancel the Mixed Media collection. Cancellation can only occur before `collectMedia`. Updates will be provided through the callbacks that are provided in the `mixedMediaListener` object.

10.2.3.1 Function Arguments

None

10.2.3.2 Return value

This function does not return anything. Instead feedback is provided through the functions defined in the *dispenseListener* argument.

10.2.3.3 Exceptions

This function will throw an exception if attempting to cancel after mixed media collection has started – ie., after the *collectMedia* operation is initiated.

10.2.4 Collect Media

```
MixedMediaService.collectMedia(): void
```

This function opens the mixed media collection device to accept a media bundle from the consumer. Updates will be provided through the callbacks that are provided in the *mixedMediaListener* object.

10.2.4.1 Function Arguments

None

10.2.4.2 Return value

This function does not return anything. Instead feedback is provided through the functions defined in the *mixedMediaListener* object.

10.2.5 Stop Collecting

```
MixedMediaService.collectMedia(): void
```

This function requests the collection of media to stop. When the collection has successfully stopped the event *mixedMediaHandler.noMediaCollected()* will be reported. If called when media has been inserted and is still being processed or the device is already stopping, this function has no effect.

10.2.5.1 Function Arguments

None

10.2.5.2 Return value

This function does not return anything. Instead feedback is provided through the functions defined in the *mixedMediaListener* object.

10.2.6 Capture Media

```
MixedMediaService.captureMedia(): void
```

This function moves all the media contained within the device into the retract location. The money still belongs to the consumer. This function is usually called after media has been inserted into the device but the consumer has walked away without completing a transaction.

10.2.6.1 Function Arguments

None

10.2.6.2 Return value

This function does not return anything. Instead feedback is provided through the functions defined in the *mixedMediaListener* object.

10.2.7 Retain Media

```
MixedMediaService.retainMedia(): void
```

This function stores all the media contained within the device escrow into the terminal safe. The money belongs to the bank. Note: if using the MixedMediaService in conjunction with a NextGen transaction Service, the transaction service will automatically call Retain Media as part of transaction completion.

10.2.7.1 Function Arguments

None

10.2.7.2 Return value

This function does not return anything. Instead feedback is provided through the functions defined in the *mixedMediaListener* object.

10.3 Deposit Service API

The Deposit Service API is exposed to allow the NextGen application to initiate a cash deposit transaction. Applications retrieve this object on start up from the NextGen Registry (“Deposit”).

The Deposit Service instance is required to expose the following functions.

- `getInfo () : DepositInfo`
- `authorize (authorizeRequest, authorizeListener) : void`
- `deposit (depositListener) : void`

The coming sections provide a detailed description of these functions, and the objects and flows that are involved.

10.3.1 Get Deposit Information

`DepositService.getInfo(): DepositInfo`

Returns information about the Deposit information for the ATM.

10.3.1.1 Function arguments

This call does not require parameters.

10.3.1.2 Return value

The object returned will be an instance of *DepositInfo* containing the following members.

Field	Type	Required	Description
<code>readyForDeposit</code>	boolean	Required	Describes whether the Deposit Service is ready for interaction. Will return false if no media was provided through the mixed media service, or when no card/account information is present within the session.

10.3.2 Authorize

`DepositService.authorize(authorizeRequest, authorizeListener): void`

Instructs the ATM to start processing a Deposit with the provided values. The ATM will process the transaction on the Network of the card that was read. Event updates will be provided through the callbacks that are provided in the *authorizeListener* object.

The amount that will be deposited depends on the media that was provided through the Mixed Media Service.

10.3.2.1 Function arguments

This function call requires the following arguments.

Argument	Type	Required	Description
<code>authorizeRequest</code>	object	Required	Object containing the deposit request
<code>account</code>	account object	Optional	Account (object) to authorize
- <code>accountNumber</code>	string		
- <code>bankId</code>	string		
- <code>otherId</code>	string		

token	string	Optional	AuthToken data to be send to the acquirer. Token may be obtained in a cardless transaction, e.g. via QRCode.
authorizeListener	object	Required	Object containing callback functions that will be used by the ATM to provide the application with feedback during the deposit process.
authorized (event)	function	Required	This function will be called when the host system has authorized the deposit.
unableToAuthorize (responseCode)	function	Required	This function will be called when the host system has not authorized the deposit. Response code indicates the cause of the failure.
error (error)			This function will be called when an error occurred while processing the deposit. Error details are provided through the <i>error</i> object.

10.3.2.2 Return value

This function does not return anything. Instead feedback is provided through the functions defined in the *authorizeListener* argument.

10.3.2.3 Exceptions

This function will throw an exception if one or more of the required arguments were not provided. This function will throw an exception if there is no card-data available within the active session. This function will throw an exception if there is no media presented through the mixed media service within the active session.

10.3.3 Deposit

```
DepositService.deposit(depositListener): void
```

Instructs the ATM to fulfil the Deposit that was processed via the *DepositService.authorize* function. Event updates will be provided through the callbacks that are provided in the *depositListener* object.

10.3.3.1 Function Arguments

This function requires the following arguments.

Argument	Type	Required	Description
depositListener	object	Required	Object containing callback functions that will be used by the API to provide asynchronous updates during the output cash process.
deposited ()	function	Required	Indicates the deposit has been finalized.
error(error)	function	Required	This function will be called when the ATM was unable to return the card. E.g. when the contactless reader was used to read the card data. Error details are provided through the <i>error</i> object.

10.3.3.2 Return value

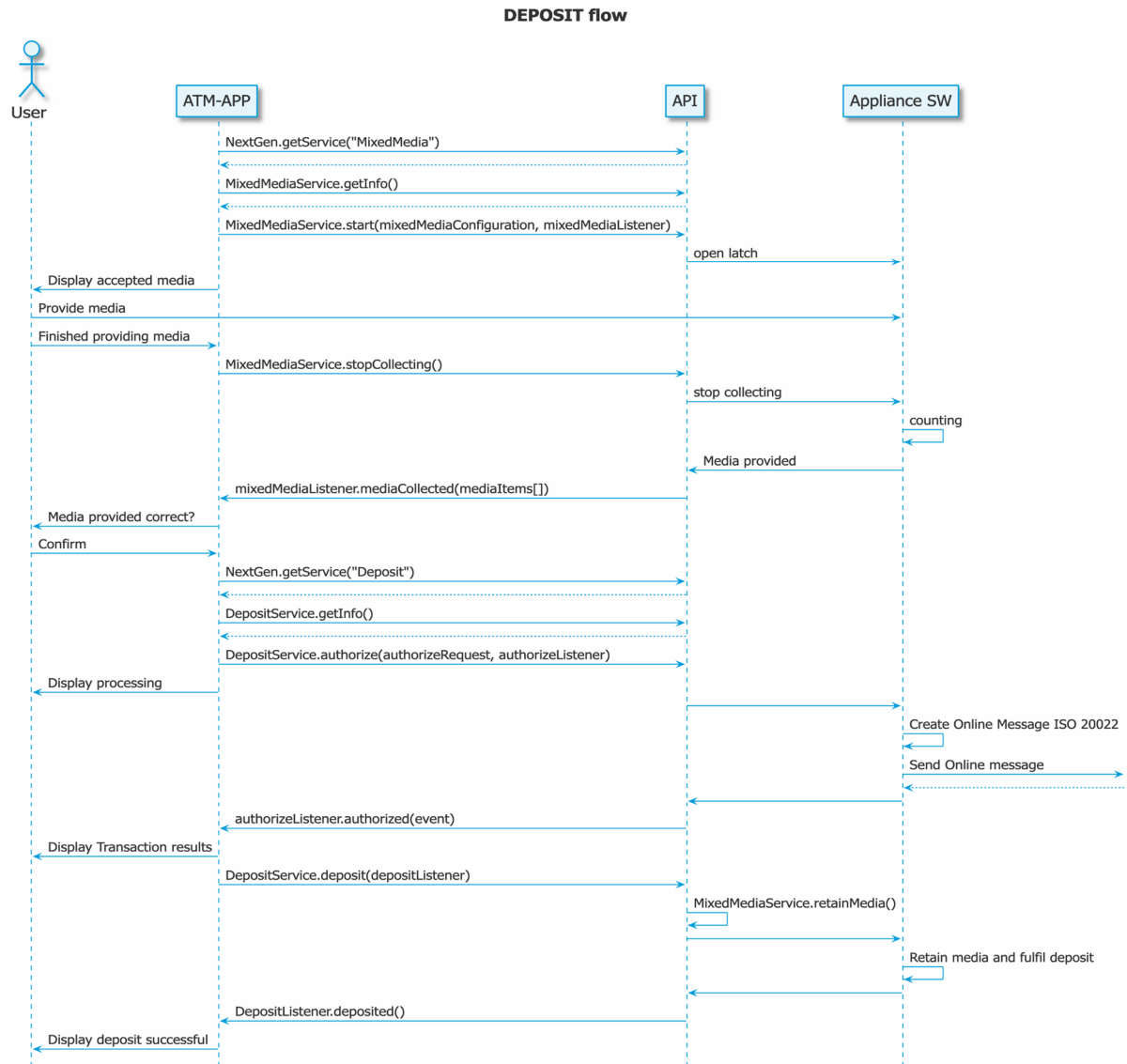
This function does not return anything. Instead feedback is provided through the functions defined in the *depositListener* argument.

10.3.3.3 Exceptions

This function will throw an exception if one or more of the required arguments were not provided. This function will throw an exception if no transaction was processed previously within the session. (*DepositService.authorize(...)* was not completed successfully.)

10.3.4 MixedMedia and Deposit Service flow

The flow below illustrates how the Mixed Media and Deposit services are supposed to work together.



10.4 CryptoSell Service API

The CryptoSell Service API is exposed to allow the NextGen application to initiate a cash withdrawal transaction from a cryptoWallet. Applications retrieve this object on start up from the NextGen Registry (“CryptoSell”).

The CryptoSell Service instance is required to expose the following functions.

- `getInfo () : CryptoSellInfo`
- `processCryptoTransaction (processCryptoRequest, transactionListener) : void`
- `dispense (dispenseListener) : void`

The coming sections provide a detailed description of these functions, and the objects and flows that are involved.

10.4.1 Get CryptoSell Information

`CryptoSellService.getInfo(): CryptoSellInfo`

Returns information about selling crypto currency to the ATM in order to withdrawal cash.

10.4.1.1 Function arguments

This call does not require parameters.

10.4.1.2 Return value

The object returned will be an instance of *CryptoSellInfo* containing the following members.

Field	Type	Required	Description
<code>available</code>	boolean	Required	Describes whether the CryptoSell Service is available and ready for interaction.
<code>denominations</code>	list of amount objects	Required	A list containing the denominations available from the ATM.
<code>currency</code>	string	Required	The currency of the amount. ISO-4217
<code>amount</code>	number	Required	The value of the amount.

10.4.2 Authorize

`CryptoSellService.processCryptoTransaction (processCryptoRequest, transactionListener): void`

Instructs the ATM to start processing a Crypto transaction with the provided values. The ATM will process the transaction with the Wallet that was provided. Event updates will be provided through the callbacks that are provided in the *authorizeListener* object.

10.4.2.1 Function arguments

This function call requires the following arguments.

Argument	Type	Required	Description
<code>processCryptoTransaction</code>	object	Required	Object containing the crypto transaction request
<code>amount</code> - <code>currency</code> - <code>value</code>	amount object string number	Required	Amount to credit from the wallet
<code>cryptoWallet</code> - <code>walletId</code> - <code>phoneNumber</code>	account object string string	Required	Wallet from which the cryptoAmount will be credited.
<code>notemix</code> - <code>count</code> - <code>denomination</code> <code>currency</code> <code>value</code>	denomination[] number amount object	Optional	Notemix of cash to dispense.
<code>transactionListener</code>	object	Required	Object containing callback functions that will be used by the ATM to provide the application with feedback during crypto transaction process.
<code>transactionInProgress(</code> <code>event</code> <code>)</code>	function	Required	This function will be called when the transaction was initiated successfully by the ATM.
<code>transactionFailed(</code> <code>responseCode</code> <code>)</code>	function	Required	This function will be called when the host system has not authorized the withdrawal. Response code indicates the cause of the failure.
<code>error (</code> <code>error</code> <code>)</code>			This function will be called when an error occurred while processing the transaction. Error details are provided through the <i>error</i> object.

10.4.2.2 Return value

This function does not return anything. Instead feedback is provided through the functions defined in the *authorizeListener* argument.

10.4.2.3 Exceptions

This function will throw an exception if one or more of the required arguments were not provided. This function will throw an exception if there is no card-data available within the active session. This function will throw an exception if the CryptoSell service is not ready for use.

10.4.3 Dispense

```
CryptoSellService.dispense( dispenseListener): void
```

Instructs the ATM to fulfil the money dispense that was requested via the *CryptoSellService.processCryptoTransaction* function. Event updates will be provided through the callbacks that are provided in the *dispenseListener* object.

10.4.3.1 Function Arguments

This function requires the following arguments.

Argument	Type	Required	Description
<code>dispenseListener</code>	object	Required	Object containing callback functions that will be used by the API to provide asynchronous updates during the output cash process.
<code>cashPresented ()</code>	function	Required	Indicates cash has been presented to the consumer
<code>cashTaken ()</code>	function	Required	Indicates cash has been taken by the consumer
<code>cashRetracted ()</code>	function	Required	Indicates cash was retracted after being presented
<code>failedRetractCash ()</code>	function	Required	Indicated a failure retracting cash
<code>error(error)</code>	function	Required	This function will be called when the ATM was unable to return the card. E.g. when the contactless reader was used to read the card data. Error details are provided through the <i>error</i> object.

10.4.3.2 Return value

This function does not return anything. Instead feedback is provided through the functions defined in the *dispenseListener* argument.

10.4.3.3 Exceptions

This function will throw an exception if one or more of the required arguments were not provided. This function will throw an exception if no transaction was processed previously within the session. (*CryptoSellService.processCryptoTransaction(...)* was not completed successfully.)

10.5 One-Time Passcode Service API

The One-Time PassCode Service API is exposed to allow the NextGen application to initiate secure OTP entry. The OTP entry is required for allowing users to authorise (crypto) transactions by means of Multi-factor authentication. Applications retrieve this object on start up from the NextGen Registry ("OTP").

The OTP Service instance is required to expose the following functions.

- `collectOTP (collectOTPListener): void`

The coming sections provide a detailed description of these functions, and the objects and flows that are involved.

10.5.1 Collect OTP

```
OTPService.collectOTP(collectOTPOptions, collectOTPListener): void
```

Requests the ATM to obtain the OTP code from the end-user. This function requires that (crypto)transaction is in progress.

10.5.1.1 Function arguments

This function requires the following arguments.

Argument	Type	Required	Description
<code>options</code>	object	Required	Object containing the collect OTP options
<code>minLength</code>	number	Optional	Minimum number of digits in the PIN. If defined must be an integer greater than 0
<code>maxLength</code>	number	Optional	Maximum number of digits in the OTP. If defined must be an integer greater than 0
<code>autoEnd</code>	boolean	Optional	Defines if OTP collection ends automatically when the length of the PIN = <code>maxLength</code>
<code>keys</code>	string[]	Optional	Array of strings that define which keys are to be enabled. <ul style="list-style-type: none"> • "enter" – terminates input • "cancel" – terminates input • "clear" – non terminating • "backspace" – non terminating • "digit" – non terminating • "00" – non terminating • "000" – non terminating • "help" – non terminating • "dec" – non terminating
<code>collectOTPListener</code>	object	Required	Object containing callback functions that will be used by the ATM to provide the application with feedback during the validate PIN process.
<code>keyPressed(key, terminated)</code>	function	Required	This function will be invoked when a key press occurs during PIN entry: <ul style="list-style-type: none"> - Key – String as defined in keys - Terminated – Boolean Indication that entry is complete
<code>error(error)</code>	function	Required	This function will be called when the ATM was unable to obtain the PIN. Error details are provided through the <code>error</code> object.

10.5.1.2 Return value

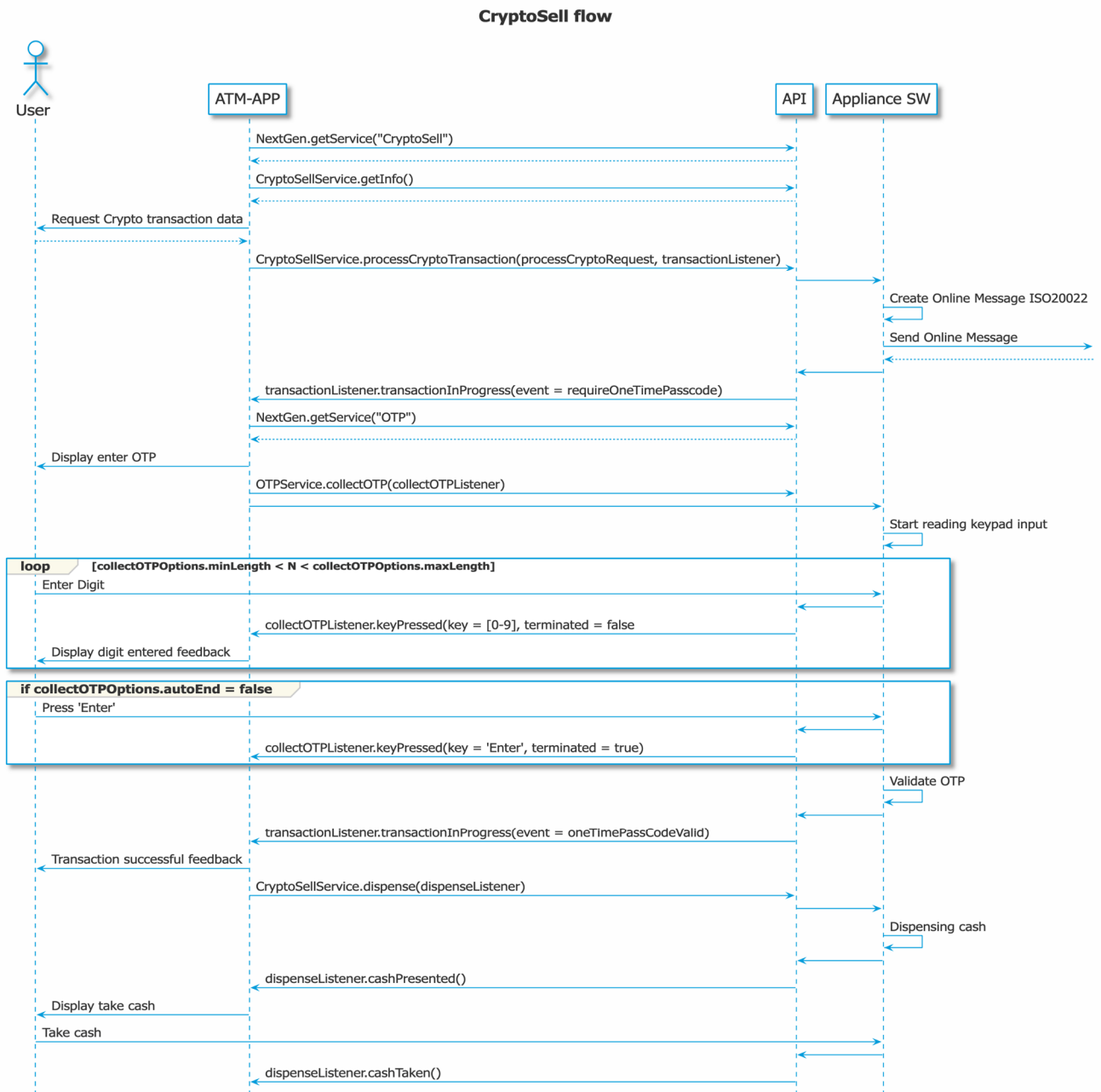
This function does not return anything. Instead feedback is provided through the functions defined in the *collectOTPListener* argument.

10.5.1.3 Exceptions

This function will throw an exception if one or more of the required arguments were not provided.

10.5.2 CryptoSell Flow

The flow below illustrates how the Crypto Sell Service, and the Collect OTP service are supposed to work together.



10.6 CryptoBuy API

The Crypto Buy API is exposed to allow the NextGen application to initiate a crypto currency purchase transaction. Applications retrieve this object on start up from the NextGen Registry ("CryptoBuy").

The CryptoBuy Service instance is required to expose the following functions.

- `getInfo () : cryptoBuyInfo`
- `authorize (processTransactionRequest, transactionListener) : void`
- `deposit (depositListener) : void`

The coming sections provide a detailed description of these functions, and the objects and flows that are involved.

10.6.1 Get CryptoBuy Information

`CryptoBuyService.getInfo(): CryptoBuyInfo`

Returns information about the Crypto Buy information for the ATM.

10.6.1.1 Function arguments

This call does not require parameters.

10.6.1.2 Return value

The object returned will be an instance of *CryptoBuyInfo* containing the following members.

Field	Type	Required	Description
<code>readyForCryptoBuy</code>	boolean	Required	Describes whether the CryptoBuy Service is ready for interaction. Will return false if no media was provided through the mixed media service, or when no wallet information is provided.

10.6.2 Authorize

`CryptoBuyService.processTransaction(processTransactionRequest, transactionListener): void`

Instructs the ATM to start processing a Crypto purchase with the provided values. The ATM will debit the deposited amount to the provided crypto wallet. Event updates will be provided through the callbacks that are provided in the *processTransactionListener* object.

The amount that will be deposited depends on the media that was provided through the Mixed Media Service.

10.6.2.1 Function arguments

This function call requires the following arguments.

Argument	Type	Required	Description
<code>processTransactionRequest</code>	object	Required	Object containing the transaction request data
<code>cryptoWallet</code> - <code>walletId</code>	account object string	Optional	Account (object) to authorize

<code>transactionListener</code>	object	Required	Object containing callback functions that will be used by the ATM to provide the application with feedback during the deposit process.
<code>transactionCompleted (event)</code>	function	Required	This function will be called when the host system has authorized the deposit.
<code>transactionFailed (responseCode)</code>	function	Required	This function will be called when the host system could not finish the crypto transaction. Response code indicates the cause of the failure.
<code>error (error)</code>			This function will be called when an error occurred while processing the deposit. Error details are provided through the <i>error</i> object.

10.6.2.2 Return value

This function does not return anything. Instead feedback is provided through the functions defined in the *authorizeListener* argument.

10.6.2.3 Exceptions

This function will throw an exception if one or more of the required arguments were not provided. This function will throw an exception if there is no media presented through the mixed media service within the active session.

10.6.3 finalize

```
CryptoBuyService.finalize(finalizeListener): void
```

Instructs the ATM to fulfil the Deposit that was processed via the *DepositService.processTransaction* function. Event updates will be provided through the callbacks that are provided in the *finalizeListener* object.

10.6.3.1 Function Arguments

This function requires the following arguments.

Argument	Type	Required	Description
<code>finalizeListener</code>	object	Required	Object containing callback functions that will be used by the API to provide asynchronous updates during transaction finalize process.
<code>finalized ()</code>	function	Required	Indicates the transaction has been finalized.
<code>error(error)</code>	function	Required	This function will be called when the ATM was unable to return the card. E.g. when the contactless reader was used to read the card data. Error details are provided through the <i>error</i> object.

10.6.3.2 Return value

This function does not return anything. Instead feedback is provided through the functions defined in the *finalizeListener* argument.

10.6.3.3 Exceptions

This function will throw an exception if one or more of the required arguments were not provided. This function will throw an exception if no transaction was processed previously within the session. (*CryptoBuyService.processTransaction(...)* was not completed successfully.)

10.6.4 CryptoBuy Flow

The flow below illustrates how the Mixed Media and Deposit services are supposed to work together.

