



SWIFTReady Access Plug-in

Label Criteria 2009

Version 2.0

January 2009

Legal Notices

Copyright

SWIFT © 2009. All rights reserved.

You may copy this publication within your organisation. Any such copy must include these legal notices.

Disclaimer

SWIFT supplies this publication for information purposes only. The information in this publication may change from time to time. You must always refer to the latest available version.

Translations

The English version of SWIFT documentation is the only official version.

Trademarks

SWIFT is the trade name of S.W.I.F.T. SCRL. The following are registered trademarks of SWIFT: SWIFT, the SWIFT logo, Sibos, SWIFTNet, SWIFTReady, and Accord. Other product, service, or company names in this publication are trade names, trademarks, or registered trademarks of their respective owners.

Table of content

1	Preface	3
	1.1 Purpose	3
	1.2 Audience	3
	1.3 SWIFTReady Programme	3
	1.4 Related Documents	3
2	Alliance plug-in	4
	2.1 ADK contents	4
	2.2 ADK usage.....	4
	2.3 ADK License	5
3	SWIFTReady Access Plug-in	6
	3.1 Scope.....	6
	3.2 Plug-in typology	7
	3.3 New requirements for 2009	7
4	Standards	8
	4.1 MT	8
	4.2 MX.....	9
5	SWIFT Messaging Protocols	11
	5.1 FIN	11
	5.2 InterAct.....	11
6	SWIFT Alliance integration	13
7	ADK Developer Guide compliance	14
8	Message Recovery	16
9	Plug-in installation and configuration	17
10	User Documentation	18
11	Plug-in Typology	19
	11.1 Message Repair.....	19
	11.2 Business Activity Monitoring (BAM).....	19
	11.3 Message filtering.....	19
12	Marketing and Sales	20

1 Preface

1.1 Purpose

SWIFT initiated the SWIFTRReady label program over 10 years ago, and maintains it to certify third-party applications and middleware products that support SWIFT messaging systems and standards and that integrate with SWIFTNet through Alliance interfaces.

This document provides a structured and detailed view of all criteria that an Alliance Access Plug-in is subject to fulfil in order to be granted the SWIFTRReady label.

Access plug-in are software components that beef up the functionalities of Alliance Access in providing the business features that are missing for now, including Anti-money laundering (AML), black-list filtering, duplicate checking, STP enrichment, business activity monitoring (BAM), and many other purposes.

1.2 Audience

The target audience for this document is both vendors considering the certification of a product, and SWIFT Users who are seeking an overview of the SWIFTRReady Label contents. The audience should be familiar with the SWIFT world from both a technical and a business perspective.

1.3 SWIFTRReady Programme

The SWIFTRReady label programme extends throughout the entire financial application chain, including Trade, Treasury, Payment, Corporate and Securities applications.

Each SWIFTRReady label defines a set of criteria, which are reviewed every year to ensure that the software remains aligned with the financial market evolution and with customer needs.

These criteria are designed to reflect the capability of a vendor product to provide message processing automation in a SWIFT context, and to support Straight through Processing (STP) in order to increase customer value, limit customisation needs and cost, and reduce time to market.

1.4 Related Documents

- [SWIFTRReady applications guide](#)
Explains the 'Why and How' on joining the SWIFT Partner Management framework and its related SWIFTRReady Accreditation programmes.
- [SWIFTRReady criteria portfolio](#)
Explains the 'What' in a generic yet detailed manner on the criteria of your SWIFTRReady Application.
- [SWIFTRReady technical validation guide](#)
Explains the 'How' in a detailed manner on how your application will be validated to become SWIFTRReady.

2 Alliance plug-in

SWIFT Alliance Access (SAA) is the most common messaging interface used by SWIFT customers to exchange FIN and ISO2022 messages. In the past years, many vendors have been developing adapters and plug-in components to enrich the features of Alliance Access.

This paper provides an overview on the Alliance Access Developer Kit (ADK) used for this purpose.

2.1 ADK contents

The ADK is a collection of software and documentation that constitutes the open interface to Alliance Access. The ADK enables third party and financial institution developers to build their own applications for interfacing with Alliance Access.

The ADK relies on some key design aspects of the SAA architecture. SAA is made out of functional components. The SAA database, installation process and process control are organized along these components. With the ADK, third party applications can smoothly integrate into SAA as a new component.

The ADK provides libraries containing APIs that can call a subset of the services provided by the SAA servers. APIs are provided in C development language. No C++ or Java versions exist at this point.

The ADK APIs offer the required robustness and security level required by Alliance kernel to protect the main functions against uncontrolled ADK applications.

2.2 ADK usage

ADK provides developers with APIS that allows intercepting messages from SAA workflow, to inspect them, and re-route them after optional modification. Messages can be captured in both direction (messages coming from back-office application and route to SWIFT, or messages coming from SWIFTNet, before they reach any business applications).

Messages are locked by the ADK component. No message can be created nor deleted by an ADK component.

The fact that messages can be intercepted just before they get out to SWIFT (even if they are created locally using SAA GUI), or before they can be processed by any business applications, makes the ADK an interesting developer tool for Anti Money Laundering (AML), anti-terrorist, OFAC, watch and caution lists, and other PEP (Politically Exposed Persons) filtering systems.

The integration with back-office applications can also be based on ADK, or to monitor the messaging flows for reporting or audit purpose, using BAM technology. Some use it to convert or to enrich messages from STP, to manage duplicates, to derive statistics, to extract messages that need extra approval before being sent out, and for many other reasons.

Note that, if most of the message access and modifications function of SAA are available for the ADK developers, some functions are disabled for security reasons. With the ADK, it is not possible to:

- modify messages that are not in the Routing Point investigated by the plug-in
- delete messages present in SAA (it is however possible to Complete messages)
- update the SAA Correspondent Information File (CIF)
- read the SAA Event Journal
- modify the System Management configuration parameters
- Authenticate or test a message

2.3 ADK License

The ADK is distributed together with the SAA software.

Two licensed packages exist for ADK:

- The **toolkit run-time** license is to be installed in each operational site where an SAA Plug-in is to run. Plug-in customers need to purchase the ADK run-time license.
- The **toolkit development** licence is required to develop ADK applications.

The runtime package consists of documentation, shared libraries that implement the ADK APIs and a specific ADK installation procedure that must be used to integrate ADK components into Alliance environments.

3 SWIFTReady Access Plug-in

3.1 Scope

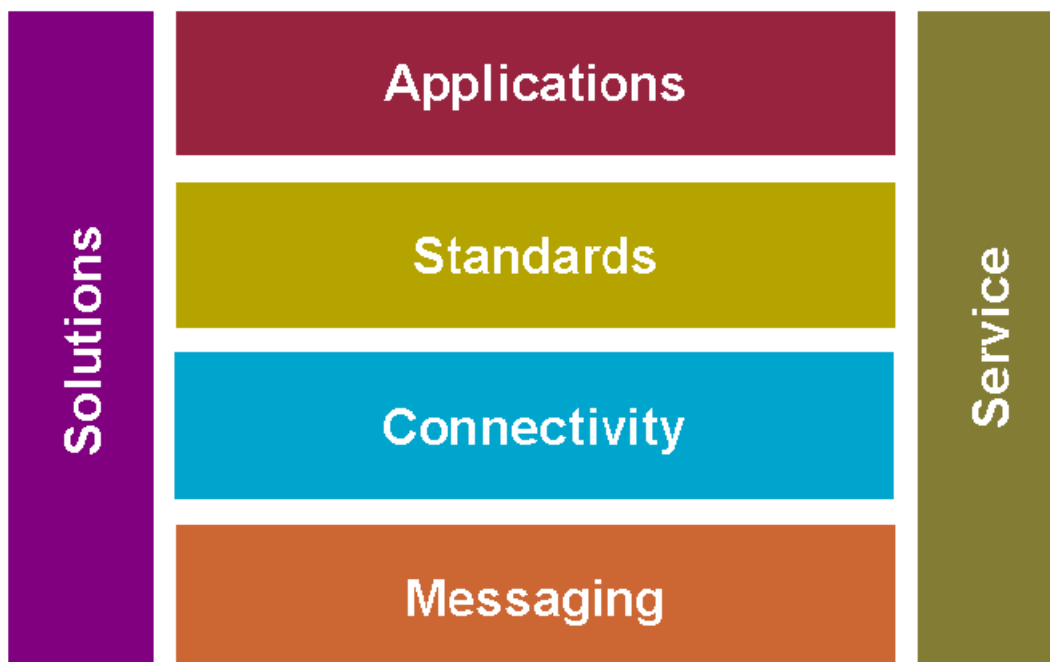
The SWIFTReady label programme was created in 1998 to ensure the proper support of SWIFT Standards, messaging services and interface connectivity by back-office applications and middleware.

A **Access Plug-in** refers to the combination of processes, software components built around SWIFT Alliance access interface, and which exhibit business or technical features that SWIFT Users are looking for, and cannot find today as part of the basic Access product.

The SWIFTAlliance Access Plug-in label is conferred to software products that have been developed with the SWIFTAlliance Access Developer Kit (ADK), and are compliant with the development guidance and good coding practices as laid down in this document

The Access Plug-in focuses on the specific needs of the financial community, including actors such as banks, brokers, fund managers, traders, banking and securities market infrastructures and corporate treasury departments.

The Access Plug-in integrates business logic with messaging services and act upon the flow of business messages which transit through SWIFT Alliance interfaces.



3.2 Plug-in typology

The Access Plug-in label intends to capture the features requested by SWIFT Users. The most common usage of plug-in can be categorised as :

1. **Filtering (AML, OFAC, ATF, Embargo, PEP)**
2. **Business Activity Monitoring (BAM)**
3. **Technical flow Monitoring**
4. **Duplicate checking**
5. **STP enhancer**
6. **Archiving**
7. **Reporting**
8. **Back-office connectivity**
9. **Network connectivity**

3.3 New requirements for 2009

To qualify for the SWIFTReady Access Plug-in label, an Access Plug-in must comply with a set of mandatory criteria which are described in the upcoming sections.

The requirements that are **new in 2009** are summarised in the below synopsis and are **marked in blue in subsequent sections**.

The last column lists all new **optional** requirements that may be supported in 2009. These will not be mandated for the conferral of the Access Plug-in label. Still, vendors will be recognised for the support of these options, whenever they can demonstrate compliance. Supported options will be published on www.swift.com, in a dedicated sheet.

Dimension	Feature	New Mandatory Requirements in 2009	New Optional Requirements in 2009
Standards	MT	SRG 2009	
	MX	SRG 2009	

The presence of optional criteria in this document provides an indication of the programme evolution, as some of the optional criteria may become mandatory in future label releases. The complete list of requirements for 2009 (new and old) are summarised in the last section of this document.

4 Standards

SWIFT develops business standards to support transactions of the financial market. FIN, based on proprietary message standards, is the original messaging service. Later, support for standards on FIN was extended to include ISO 15022 standards and the service name was changed to FIN. Message types that operate on FIN are referred to as **MT**.

In 1997 SWIFT launched SWIFTNet, an IP-based secure network. This network allows the exchange of message types represented in XML format. It provides support for message standards ISO 20022 and for FpML (for more information, please refer respectively to www.iso20022.org and www.isda.org) Messages Types of standard ISO 20022 are referred to as **MX**.

Today the network traffic of MT still represents the majority of SWIFT traffic. These are being progressively complemented and/or replaced by XML-based messages, which ease validation, and enable the transfer of richer data for complex transactions.

Label Requirement	Reference number 1	Mandatory
The Access Plug-in should support the MT and MX related to the solution at stake, as listed in SWIFT User Handbook (UHB). Message support implies the capacity to capture messages for many category and type, treat them as appropriate, and route them within Access, accordingly to the plug-in treatment result.		

4.1 MT

The 240+ FIN messages convey business payloads named Message Type (**MT**). Each MT is defined by a three-digit number and holds a specific business meaning. MTs are categorised as follows:

Category	Category Name	Number of Messages
0	System messages	
1	Customer Payments and Cheques	18
2	Financial Institution Transfers	17
3	Treasury - Foreign Exchange, Money Markets & Derivatives	27
4	Collections & Cash Letters	18
5	Securities	67
6	Treasury – Precious Metals & Syndications	20
7	Documentary Credits & Guarantees	29
8	Travelers Cheques	18
9	Cash Management & Customer Status	29

MT Messages are gathered into Business Transactions (Trading, Settlement, Funds). For instance, a Trading Transaction includes Order to Buy (MT502), Trade Confirmation (MT515, MT518), or Statement messages (MT576). More information over supported Business Transactions can be found on http://www.swift.com/index.cfm?item_id=60000.

The Access Plug-in should support the MT messages required for the plug-in Application domain. It means that they should be able to acquire messages, access the meaningful data for checking and releasing the messages as necessary.

Label Requirement	Reference number 2	Mandatory
The Access Plug-in should support all the MT messages pertaining to the solution at stake		

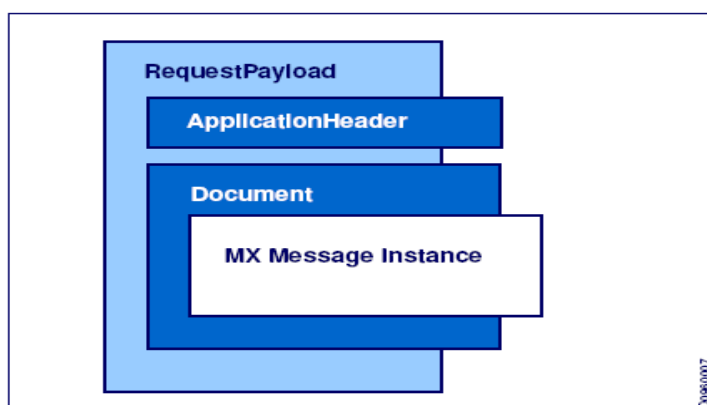
4.2 MX

Leveraging the emergence of XML as de facto standards for inter-systems communication, SWIFT uses the UNIFI (ISO20022) methodology to design standards, based on business processing modelling. The UNIFI methodology uses a central data dictionary containing reusable business elements to build XML standards, named MX, that are used in business transactions and provide interoperability across financial services.

Ultimately, the whole financial community will move to MX, but adoption will vary by business area, depending on the drivers. SWIFT intends to provide translation rules to support interoperability in business areas where coexistence of MT and MX is necessary. First translation rules for Funds and Payments have been published on www.swift.com/standards.

An MX message contains the business area specific payload. It has the structure defined by the corresponding XML Schema, as published in the UHB. These schemas provide not only message structure but also on message scope and usage, rules and guidelines. Sample data is also provided in the UHB.

The MX message is wrapped as the RequestPayload within the XML envelope. This request payload contains also the ApplicationHeader. This application header contains general information; its usage is specific to the context of the service.



The 250+ available MX messages are structured according to market segments. The *Business Area*, is represented by four (rather meaningful) letters. For example, '*camt*' stands for the 'Cash Management' Business Area. The table below gives an overview of the most common MX Business Areas.

Market	Segment	Business area code	Nb of Msg
Payment	Payment initiation	pain	5
	Payments Clearing and Settlement	pacs	7
	Exceptions and Investigations	exin	14
	Cash Management	camt	58
Trade	Trade Service Initiation	trsi	50
	Trade Services	trse	
	Trade Services Management	trsm	

Market	Segment	Business area code	Nb of Msg
Securities	Securities Trade	setr	30
	Trade Initiation	seti	32
	Settlement	sese	16
	Reference Data	reda	3
	Issuers Agents Communication		22
	Account Management		5
	Securities Management	semt	11
FXMM and Derivatives	Proxy Voting	sepv	8
	Derivatives	deri	7
	Derivatives Initiation	dein	4
Treasury	Derivatives Management	demt	4
	Treasury Initiation	trin	13
	Treasury	trea	
Treasury Management	trmt		

MX supports XML features, such as the facets on simple data type (pattern, min and max length, numerical limits, enumeration, max digit and decimal numbers). MX also supports complex data types to gather common characteristics that are inherited by XML schemas and XML instances.

Schema validation involves syntax checking against well-formed XML rules (single-rooted paired elements, opening and closing tags). In addition the XML instance must be validated against its corresponding schema (XSD), including permitted character sets, format, cardinality, enumeration, and mandatory/optional presence. XSD are available in deployment packages on swift.com/support.

Extended validation ensures that the message is validated against the MX rule Book. It involves cross-element (if field A is present then field B must take value X or Y), intra-element (the fractional part of an amount is checked against the currency), calculations (check digit of IBAN code) and external table look-up (BIC, Country or Currency Code).

Label Requirement	Reference number 3	Mandatory
The Access Plug-in should support the MX required for the application domain, as listed in SWIFT User Handbook (UHB). In particular, filtering plug-in component should be able to deal with the payment related messages (pain, pacs) as requested by the business.		

5 SWIFT Messaging Protocols

SWIFT provides several messaging services to support the needs of the financial community. FIN and InterAct cater for the automation of structured or important financial information transfers to business partners, while FileAct and Mail convey large payload of free format and bulked/low value information.

5.1 FIN

FIN is a secure, reliable and resilient, access-controlled, structured, store-and-forward messaging service. Value added processing includes message validation against SWIFT Standards, delivery monitoring and prioritisation, on top of central message storage and retrieval.

FIN supports more than 240 Message Types (MT) categorised into 9 market segments according to their business usage (Payment, Treasury, FXMM, Derivatives, Collections, Securities, Trade, Precious Metals and Cash Management).

FIN messages are made of data blocks for addressing and control (block 1 to 3), MT business payload (block 4) and system trailers block (block 5).

{1: BASIC HEADER BLOCK}
{2: APPLICATION HEADER BLOCK}
{3: USER HEADER BLOCK}
{4: TEXT BLOCK}
{5: TRAILER BLOCK}

FIN messages need to be either accepted or rejected by the receiving application by sending an MT of type F21 to the emitting institution.

Label Requirement	Reference number 4	Mandatory
The Access Plug-in should support FIN protocol. In particular, it should be able to generate the correct FIN headers, body and trailer blocks, and should be able to parse and act upon incoming messages as appropriate.		

5.2 InterAct

InterAct caters for the interactive real-time and store-and-forward exchange of messages between applications. InterAct is widely used in many of SWIFT's solutions, including Funds, Exceptions and Investigations, TSU, Proxy Voting, FpML and Cash Reporting. It is also used for for Market Initiatives such as TARGET2, SEPA, MiFID, Euroclear CCI and Giovannini.

InterAct supports XML format and other structured formats (FIX, FpML) wrapped into an XML envelope. InterAct Central Services provide enhanced validation services over MX messages, which are XML messages designed with the ISO20022 – UNIFI methodology, and FpML messages.

InterAct can be used to send and receive business payload in real-time (RT) or Store&Forward (SF) modes.

InterAct messages can be sent through Alliance Access or Alliance Gateway, using one of the available adapters. SAA accepts any XML format, including MX and FpML.

SAA provides a resilient store-and-forward local messaging hub, manages the SWIFTNet protocol (security, network and SF queues management) and reconciliation process. These features are not natively supported on SAG.

The Access Plug-in should support all InterAct modes to cover the needs of all of SWIFT's solutions and Market Initiatives. The Messaging Operations guide (see [UHB](#)) should be strictly adhered to.

Label Requirement	Reference number 5	Mandatory
The Access Plug-in should support the InterAct RT / SF protocol and adhere to the latest release of the Messaging Operations Guide in the online UHB.		

6 SWIFT Alliance integration

The Access Plug-in is tightly integrated with Alliance access using the APIs that are made available to the developer either the Alliance Develop toolkit (ADK). The ADK is a collection of software and documentation that allows tight integration with the SAA workflow engine (routing points, journalizing, and interventions history).

The ADK provides API libraries that can call a subset of the services provided by the Alliance servers both for MT and MX messages.

Label Requirement	Reference number 6	Mandatory
<p>The architectural design shall be provided to SWIFT for review and advice.</p> <p>The requirements should be assessed during design and development phases. SWIFT will verify some of these requirements in the product source code and documentation during the formal presentation of vendor's product at SWIFT HQ premises for the plug-in validation.</p> <p>The product must be compliant with the last versions of SWIFTAlliance Access software, and shall follow the release cycle of SWIFTAlliance Access. In particular, the product shall be at least re-linked, re-packaged and re-tested with every new release (major or not) of SWIFTAlliance Access software.</p>		

SAA R6.2 extends the scope of these adapters to support any XML messages using UTF-8 encoding.

Label Requirement	Reference number 7	Mandatory
<p>The Access Plug-in should support SAA R6.2.</p>		

SAA R6.3, available Q1 2009 introduces new adapters and integration means :

1. [SOAP adapter](#) for Interact messages
2. [File Transfer adapter](#) for FileAct Store&Forward (SF) support
3. [Web services](#) providing access to internal Access information. The info provided in R6.3 concerns RMA records. Further SAA release will percolate information regarding message status, events and Access management information.

Label Requirement	Reference number 8	Optional
<p>The Access Plug-in can support SAA R6.3.</p>		

7 ADK Developer Guide compliance

A SWIFTAlliance Access Plug-in product must support all the Mandatory features referred to in the ADK documentation and may support any of the additional Optional features.

The following is a non-exhaustive list of ADK Developer and Reference guides' compliance statements

1	The design and software code shall be compliant with the SWIFTAlliance and ADK conventions on ADK coding
2	The plug-in component names shall be four capital characters long and must end either with 'S' (server) or 'A' (application with GUI). Several components can co-exist within the same product.
3	The product shall be designed as to make proper usage of all necessary ADK APIs (Access control, event journal, message files, routing, security definition, process control).
4	A registration program shall be provided to register plug-in component to SAA as an integrated part of the installation process.
5	Component directory structure shall be defined in accordance with developer guide, to allow the application to run in a operational Alliance site (e.g. bin, \$ARCH), and to build correct ADK Installation media
6	It is recommended to store Configuration files under the <i>ADK_DIR</i> sub-directory of the SWIFTAlliance Access database directory, and Data files under a subdirectory named after the plug-in component name. This sub-directory should not contain any trace files.
7	Program and executable names shall be prefixed by the plug-in component name
8	The plug-in component data files shall be different from the ones used by SWIFTAlliance and ADK
9	Reserved symbol names used by SWIFTAlliance shall not be used, as this can lead to unexpected behaviour of the faulty program even if its compilation succeeded. A list of all the reserved symbol names is provided in Developer Guide Appendix A.
10	All entity variables used in programs shall be initialised before being used.
11	Automatic re-launch after crash shall be configured via the fields <i>exsa_num_retries</i> and <i>exsa_retry_period</i>
12	All plug-in processes shall register themselves with SWIFTAlliance Access using the adequate API <i>PcsRegister</i> . A registration script should be provided for this purpose.
13	Significant events related to the start/stop of the plug-in and any message processing shall be journalised in standardised format at a centralised location known as the Event Journal.
14	The journalisation of an event may involve informing users through an alarm in the form of a window that pop-up on specified operator screens to display alarm information. Journal event have a severity. Information and warnings events should not be defined as alarms. Severe and fatal errors should be defined as alarms
15	Journal events have a class. At least the following should be used: <ul style="list-style-type: none"> - Process: for process start/stop and general information - Message: for message processing - Communication: if the component provide a link to an external system - Data: for internal data handling, like configuration files

16	At installation time each application component with a GUI shall register an <i>appl</i> record that will define its icon (as described in Developer Guide's Process Control section).
17	If a GUI needs to exhibit usage restrictions, it shall register a <i>func</i> record for every restriction function. If the functions need parameter values then a <i>perm</i> record must be registered for every permission
18	All ADK APIs return codes shall be trapped and appropriate message shall be logged in the event journal or in the trace file.
19	Interventions shall be written in the message history each time a significant event happens to the message (message checked, modified, approved...)
20	A message appendix shall be created when a message enters or leave SAA through the component (i.e. link to an external system)
21	In case a message is modified (Message enrich/repair plug-in), the original text shall be kept in a free text intervention and clear indication on what has been changed/added/removed.
22	The plug-in shall follow the SWIFTAlliance Access release cycle. Partners shall at least re-link, re-package and re-test the plug-in product within the three months of reception of a new SWIFTAlliance Access release. Form time to time, code updates shall be performed as indicated in SWIFTAlliance release letter. This holds for major and minor SWIFTAlliance Access releases.

Label Requirement	Reference number 9	Mandatory
The ADK component should be compliant with ADK Developer and Reference Guides.		

8 Message Recovery

1	Processes using API to update the database shall have a recovery process available to deal with API failure (return code ADK_FAILURE).
2	Message Processing Functions (MPFN) shall be designed with the assumption that it is potentially being started after the crash of an earlier MPFN process. MPFN should <i>first</i> do its recovery and afterwards do its normal processing. Alternatively, an ADK component can decide to have an ADK exec to perform the recovery.
3	The recovery process shall check the status of reserved messages, and perform a undo or completion of the message depending on the message status
4	If at any stage of processing, a message processing API returns ADK_FAILURE, then the MPFN shall <i>exit</i> with the failure code ADK_FAILURE_EXIT. The SAA control software should then take care of restarting the MPFN, whenever this is possible.
5	If a MPFN makes more than one update to a message instance then after each update the message shall be in a <i>recognisably different state</i> (state held in SAA database that the recovery process is able to detect) using for instance an appendix or an intervention, so that the recovery process can know how much of the normal processing has been performed and possibly activate a roll-back or a process completion.
6	The recovery process shall either <i>undo</i> (roll-back) what was done by the normal processing or can <i>finish</i> what was left unfinished. In particular, every message shall be unreserved at the end of the recovery process.

Label Requirement	Reference number 10	Mandatory
The ADK component should develop recovery procedure to ensure that no message got lost or result in an unknown status as a follow up of Access failure.		

9 Plug-in installation and configuration

The different plug-in components to be installed and registered with the ADK plug-in should be clearly described and provided to SWIFT, in order to facilitate customer support.

1	ADK Server (Component name (4 capital letters). Server names should ends with an "S".
2	ADK Client Component name (4 capital letters). Client names should end with an "A".
3	Message Processing Function (MPF) - Each <i>mpfn</i> should have a process (<i>exsa</i>), a processing (<i>mpfn</i>), routing points (<i>rpoi</i>) and an application (<i>appli</i>) records.
4	Event Messages Journal entries should all start with the component names. This allows easy tracking of plug-in behavior in isolation.
5	Routing Points - Routing points are logical locations where message instances are located and processed.
6	Registration program - Registration program should be written in TCL (Tool Command Language).
7	Configuration Parameters – each <i>cnfg</i> parameter name should be provided
8	Functions - functions (<i>func</i>) names.
9	Application - each <i>appl</i> parameter name should be provided
10	Permissions - permissions (<i>perm</i>) names.

Label Requirement	Reference number 11	Mandatory
The ADK components should be able clearly described and provided to SWIFT, in order to facilitate customer support.		

10 User Documentation

A User and installation manual shall be provided together with the Plug-in product. The following information shall be made available in the product documentation. The aim is to ensure smooth installation, configuration and daily usage at customer premises.

1	Installation of server and application components on each operating system
2	Component registration using cipher password
3	List of installed entities (components, routing points, routing rules, configuration parameters)
4	Typical product configuration (routing rules and schema), when plug-in can be configured.
5	Create, Approve and Activate new schema using Routing Application in SA Workstation (when appropriate)
6	Refine product specific routing rules using SAA Routing application. Customize routing points thresholds and routing rules (sequence numbers, routing conditions and results)
7	Modify default plug-in parameters using system Management (sleep time, in and out directories)
8	List journal entries (number, classification, text and explanation)

Label Requirement	Reference number 12	Mandatory
A complete User and installation manual should be provided to SWIFT together with every Plug-in product		

11 Plug-in Typology

11.1 Message Repair

The Access Plug-in allows editing messages in order to repair them in exceptional cases. This holds for messages that do not pass validation and need urgent repair prior to sending them to SWIFTNet.

Messages manipulation can be restricted to the user with dedicated security profile. Actions taken on messages will be logged, and messages will be stored before and after manipulation for audit purposes.

The message editor will be format aware. In particular, field names will be expanded, and field options will be provided depending on expected type (ex: code words, dates, BIC look-up).

11.2 Business Activity Monitoring (BAM)

BAM tracks processing chains at both technical and business levels. This dual competence helps financial institutions to maintain control of their data flows, through detailed views of inter-application data transformation. Through multiple connectors positioned at all levels of the processing chain, BAM ensures end-to-end supervision and monitoring, guaranteeing complete control of all data flows.

A Business Transaction is made of messages flowing from back-office to SWIFT (ex: Payment initiation) and back from SWIFT to the Back-office (ex: Payment confirmation). Message events (message generated, validated, repaired, sent, acknowledged, delivered, reconciled) can be tracked and displayed in the dashboard for monitoring purposes. Rules can be generated to associate alarms to some events (ex: message DeIN note received after 3 days). Reports should be generated.

11.3 Message filtering

AML refers to the legal controls that require financial institutions and other regulated entities to prevent or report money laundering activities.

Today, all financial institutions globally are required to monitor, investigate and report transactions of a suspicious nature to the financial intelligence unit of the central bank in the respective country. For example, a bank must perform due diligence by having proof of a customer's identity and that the use, source and destination of funds do not involve money laundering.

Anti-Terrorist Financing (ATF) is another filtering component, classified under the Suspicious Activity Reports (SAR).

The Office of Foreign Assets Control (OFAC) is a US agency that enforces economic and trade sanctions against targeted foreign states, organizations, and individuals.

Plug-in component extract messages from the SAA flows, and compare message data against check lists. Suspected messages can be disposed to a security officer through a GUI, or routed to a "suspicious queue" for further validation.

12 Marketing and Sales

Collaboration in terms of administrative and marketing information is requested. In particular the Application Provider should provide SWIFT under non-disclosure agreement with customer related information.

The application provider should provide SWIFT with a list of at least three live customers that actively use the Access Plug-in to connect to SWIFT.

By 'customer' we mean separate financial institutions using the product to generate and receive messages and files transported by SWIFT. Five live sites from the same customer are not enough to qualify for a label.

SWIFT reserves the right to contact the relevant customer to validate the functionality of the application submitted for SWIFTReady certification. A questionnaire will be sent as the basis for the customer validation which can be in the form of a telephone interview, an e-mail or a discussion at the customer site.

The information provided by the customer will be treated as confidential and will not be disclosed, unless explicitly expressed by the customer.

Label Requirement	Reference number 13	Mandatory
<p>The application provider should supply the following information under NDA:</p> <ul style="list-style-type: none"> <input type="checkbox"/> A list of at least five live customers that actively use the Access Plug-in in a SWIFT context. <input type="checkbox"/> A list of all customers active in the finance sector. The list should provide institution names, locations, and an overview of the integration scope (domain, features, and sites) for the present and previous year. <input type="checkbox"/> A product roadmap for 2009 and 2010 containing the plans for further Access Plug-in development, support of SWIFT's solutions and new releases. <input type="checkbox"/> A complete set of Access Plug-in documentation, including features overview, SWIFT adapters, workflow engine capability and user manuals. <input type="checkbox"/> A dedicated web page on Vendor web site describing the SWIFTReady application used in a SWIFT context. The page should be maintained with the same URL for a complete year that will be referenced to on www.swift.com 		

-end of document-