Swift Compatible Interface

FileAct Real-time Interface Conformance Statement

BOX Messaging Hub

This document lists the mandatory and optional requirements supported by the FileAct Real-time messaging interface for use with SwiftNet 7.x

June 2026

Table of Contents

1 General Information **3**

1.1 Interface Provider 3

1.2 Product information 3

1.3 Operational environment 3

1.4 Customer implementation environment 3

1.5 Packaging statement 4

1.6 Integration support 4

2 Conformance requirements **5**

2.1 Vendor messaging interface local features 5

2.2 SwiftNet generic features 8

2.3 Real-time FileAct features 10

1 General Information

1.1 Interface Provider

Full name of the organisation that has registered this interface product and the name of the author of this conformance statement.

Organisation	InterCope GmbH
Partner Identifier Code (PIC)	PTSADESS
Author	Andrea Lelli
Date	2011-present (Renewal 2026)

1.2 Product information

The name and version numbers of the interface product to which this compliance validation and conformance claim applies.

Product name	BOX Messaging Hub
Product version	3.27.2

1.3 Operational environment

The hardware platform(s) and/or software platforms for which this product's performance is guaranteed.

Hardware platform on which product is guaranteed	BM System z IBM p_series PC
Software platform on which product is guaranteed	AIX Windows Linux z/Linux z/OS_USS

1.4 Customer implementation environment

The hardware platform and software environment in which this interface product's customer implementation is defined (as required to achieve full compliance after an interim compliance).

Hardware platform on which product was implemented	BM System z IBM p_series PC
Software platform on which product was implemented	AIX Windows Linux z/Linux z/OS_USS

1.5 Packaging statement

The main possibilities are:

- The product is a vendor messaging interface only if the main purpose is to exchange messages or files between business applications and Swift.
- The product is integrated with another if the product offers other functionality such as connectivity to other external networks or the product is also a business application that creates and processes messages and files.
- The communication interface used by the product.
- The security administration interface used by the product. For example, the management of security endpoints and its roles by security officers can be done by the product itself and/or by using SwiftNet Online Operations Manager or another product.

Other variations are possible. If used, these should be described below.

Product is a vendor messaging interface only	No
Product is integrated with another (specify)	Messaging Interface is part of BOX Messaging Hub
Communication Interface	Alliance Gateway
Security Administration	Alliance Web Platform
Other	BOX Messaging Hub includes Backoffice integration capabilities

Note Relationship Management (RMA) is now managed through the RMA Portal since March 2024.

1.6 Integration support

The table describes if the product uses the Message Queue Host Adapter or Remote API Host Adapter as specified by Swift, or if it uses a proprietary or other industry standard solution.

Products	
MQHA	Yes
RAHA (SAG APIs)	Yes
Other	Internal Interface/Protocol to BOX Messaging Hub

Note SNL APIs (in C language) are no longer supported as from RAHA 7.7. However, RAHA 7.6 is compatible with Alliance Gateway 7.7

2 Conformance requirements

The conformance requirements for a real-time FileAct messaging interface for SwiftNet release 7.0 are specified in the corresponding interface specifications. A real-time FileAct messaging interface for SwiftNet release 7.0 must support the mandatory items referred to in the messaging interface specifications and any of the additional optional items.

The tables below identify the mandatory and optional elements that a real-time FileAct messaging interface product may support.

- Column 1 identifies the feature.
- Column 2 contains references to notes which describe the feature in more detail and where appropriate gives reference to the specification source.
- Column 3 describes whether the feature is Mandatory (M) or Optional (O) or mandatory if Supported (S).
  - A Mandatory feature must be available for all users of the product.
  - An Optional feature, if implemented, is also subject to compliance validation.
  - A mandatory if Supported feature must be available if this role is supported by the vendor messaging interface (typically related to copy-destination)
- Column 4 should be ticked “Y” or “N” to indicate support of the feature.

2.1 Vendor messaging interface local features

This section lists the features that are applicable to the vendor messaging interfaces as an application.

2.1.1 Security features (SEC)

The ability of the vendor messaging interface to manage the security related to connected operators, connected applications and SwiftNet certificate management.

Feature	Note	M/O	Sup
Access control	SEC.1	M	Y
Audit log	SEC.2	M	Y
Security officer	SEC.3	O	Y
Access control using 4-eyes principle	SEC.4	O	Y
Local authentication (LAU) between communication interface and vendor messaging interface if on different hosts for file exchanges	SEC.5	O	Y
Local authentication (LAU) between vendor messaging interface and business applications if on different hosts for file exchanges	SEC.6	O	Y
Usage of SwiftNet Link security contexts	SEC.7	M	Y
Renew rarely used SwiftNet Link security contexts	SEC.8	M	Y

Note

SEC.1 Access to the vendor messaging interface must be controlled. Updates in configuration information must be done by a user identified in a login process by providing a username and password or equivalent methods.

SEC.2 The audit log allows analysis of who changed what when. It also logs attempts to login by a user.

SEC.3 Special users that can access and update security related information. Examples of security related updates are:

- Adding, deleting, modifying users
- Resetting passwords

SEC.4 Two different users must agree with a change before it is applied.

SEC.5 This applies to the protocol between communication interface and vendor messaging interface. The Local Authentication Requirement Specification provides more information. If the vendor messaging interface manages SwiftNet PKI security contexts, LAU is optional. If it relies on the communication interface to open security contexts, LAU is mandatory.

SEC.6 This applies to the protocol between vendor messaging interface and business applications.

SEC.7 SwiftNet Link security contexts must be useable by entitled entities only. The implementation depends on the features offered by the communication interface.

SEC.8 If a certificate is not used regularly, there is a risk that it will become invalid or expire. Once invalid or expired, the certificate will no longer be able to be renewed and must be recovered.

2.1.2 Applications support

The ability of the vendor messaging interface to interact with back-office applications.

Feature	Note	M/O	Sup
Route incoming traffic to the correct business application	AS.1	M	Y
Forward received signatures	AS.2	O	Y
Forward own signatures	AS.3	O	Y
Availability of vendor messaging interface without connectivity to Swift	AS.4	O	Y
Provide vendor messaging interface processing information	AS.5	M	Y
Provide SwiftNet processing information	AS.6	O	Y

Note

AS.1 This routing can be based on various parameters taken from the received data. At a minimum, routing must be possible on the `Service` and `RequestType` taken from the `RequestHeader` or `FileRequestHeader` information.

AS.2 The signatures on data received can be made available to business applications requiring them.

AS.3 Ability to request a return of signatures on data sent and making them available to business applications requiring them.

AS.4 This feature allows business applications to send and receive files even if the vendor messaging interface is not connected via the communication interface to Swift. The vendor messaging interface is a kind of hub between the business application and Swift.

AS.5 The most important processing information that can be passed consists of the verification result of the signatures. The minimum requirement is to allow routing the file requests based on the verification result of the signature.

AS.6 The processing information is related to non-repudiation, references added by Swift, routing information, copy related information such as the copy status. The business application can receive all information or a configured subset of processing information to be received.

2.1.3 Operational features

The functionalities of the messaging interface related to the operational actions.

Feature	Note	M/O	Sup
Traffic logging	OF.1	O	Y
Unattended operations	OF.2	O	Y
Backup/restore of messaging information	OF.3	O	Y
Backup/restore of configuration data	OF.4	O	Y

Note

OF.1 Separate log from the actual requests sent or received is available for event analysis.

OF.2 The ability to use a vendor messaging interface with minimal operator intervention.

OF.3 The ability to backup and restore messaging data (files).

OF.4 The ability to backup configuration data. Depending on the design it can be several.

2.1.4 Application service profile

The ability of the vendor messaging interface to load and use the SwiftNet service definition file.

Feature	Note	M/O	Sup
Application Service Profile Package Import	ASP.1	M	Y
Application Service Profile Package Usage	ASP.2	M	Y

Note

ASP.1 The vendor messaging interface can import the package and apply the definitions of the application service profiles.

ASP.2 An application service profile contains a set of parameters as decided by the Service Administrator during the definition of the service. The application service profile is used by vendor messaging interfaces and applications to correctly send and receive traffic for that service.

2.1.5 Relationship management

The ability of the vendor messaging interface to filter traffic based on RMA authorisations.

Note that as an alternative, API calls are also available to query the RMA Portal for authorisation statuses.

Feature	Note	M/O	Sup
Check RMA authorisation to send	RM.1	O	Y
Check RMA authorisation to receive	RM.2	O	Y
RMA deployment – `RMAChecked` indication	RM.3	O	Y
Local check mode in RMA trial period configuration	RM.4	O	Y
RMA deployment reporting	RM.5	O	Y
RMA distribution file import	RM.6	O	Y

Note

RM.1 The vendor messaging interface checks the RMA authorisation before sending FileAct transfer request. Traffic that fails is made available for manual investigation by an entitled user.

RM.2 The vendor messaging interface checks the RMA authorisation after receiving FileAct transfer request. Traffic that fails is made available for manual investigation by an entitled user.

RM.3 The vendor messaging interface indicates the usage of the authorisation-to-send in the `RequestControl` as appropriate.

RM.4 This local configuration changes the behaviour of the checking of RMA authorisations-to-send and authorisations-to-receive during the Trial period.

When check mode is on, then when a check fails, the traffic is stopped.

When check mode is off, then when a check fails the traffic is not stopped.

RMA.5 The vendor messaging interface provides information about the usage of authorisations for traffic sent and received.

RMA.6 The vendor messaging interface can import and process an RMA partial and complete distrinution file received from the RMA Portal.

The report can be integrated within traffic investigation reports or can be integrated within audit log reports.

2.2 SwiftNet generic features

This section covers generic processing of SwiftNet primitives across all SwiftNet protocols.

2.2.1 Product information (PRIN)

The vendor messaging interface must identify himself in every XML primitive.

Feature	Note	M/O	Sup
Vendor messaging interface identification within `ProductList`	PRIN.1	M	Y
Application identification within `ProductList`	PRIN.2	M	Y

Note

PRIN.1 The vendor messaging interface identifies itself in the `ProductList`.

PRIN.2 It also provides the ability for registered applications to use the `ProductList` within file transfer request created by those applications or the vendor messaging interface adds the `ProductList` when it identifies that an application is connected to the vendor messaging interface.

2.2.2 PKI signature (SIGN)

The ability of the vendor messaging interface to sign SwiftNet primitives.

Feature	Note	M/O	Sup
SwiftNet Signature processing	SIGN.1	M	Y
User Digital Signature processing	SIGN.2	O	Y

Note

SIGN.1 The vendor messaging interface must properly sign traffic it sends to Swift. Properly signing means to select the signature format (`Crypto` or `SignatureList`), and to select what is signed (what `DigestRef` to add). What is signed depends on the service and request type.

SIGN.2 The vendor messaging interface may offer functions to use SwiftNet PKI certificates for digital signature within the ISO20022 Business Application Header (BAH or head.001.001.01) or within other elements such as the `Xchg` element (Business File Header (BFH) or head.002.001.01) used by T2S.

2.2.3 Non-repudiation (NR)

The ability of the vendor messaging interface to support non-repudiation of emission and reception.

Feature	Note	M/O	Sup
Non-repudiation support	NR.1	M	Y

Note

NR.1 Non-repudiation can be mandatory or optional for a service or for a set of request types.

2.2.4 Duplicate file transfer request (DUP)

The ability of the vendor messaging interface to flag XML primitives as duplicated and to process received duplicated primitives.

Feature	Note	M/O	Sup
Using E2EControl for PDE	DUP.1	M	Y

Note

DUP.1 `E2EControl` must be used to identify the file transfer request for which a possible duplicate information is to be provided. The vendor messaging interface must add an `E2EControl` if not provided by the business application generating the file transfer request.
A specific test involving back-office possible duplicate without E2EControl is optional (RT_FA_DUP_03) If the file download functionality is not supported, related duplicate tests are not applicable.

2.2.5 Client and server functionality (SDRC)

The ability of the vendor messaging interface to process sent and received SwiftNet primitives in parallel.

Feature	Note	M/O	Sup
Provide client and server functionality	SDRC.1	M	Y

Note

SDRC.1 The client and server primitives from the communication interface must be used so that the vendor messaging interface can play the role of client and of server in an efficient way.
This requires following the order of primitives to be sent as explained in SwiftNet Service Design Guide and depends on the features offered by the communication interface.

2.2.6 HeaderInfo support (HDI)

The ability of the vendor messaging interface to summarise a file content into an enhanced header and to use such a received header.

Feature	Note	M/O	Sup
Using `HeaderInfo` for file information	HDI.1	M	Y

Note

HDI.1 The `HeaderInfo` summarises the file content allowing the messaging interface to take business decision without parsing the entire file. This is mandatory for services like bulk payments and FileAct copy.

2.2.7 Error management (ERR)

Feature	Note	M/O	Sup
Usage of enhanced errors	ERR.1	O	Y
Use of `CallbackOnError` functionality	ERR.2	O	N

Note

ERR.1 Enhanced errors are used by the vendor messaging interface when the `Sw:ErrorCodeMode` is set appropriately. The New errors will be mandated for SwiftNet 8.0

ERR.2 The `CallbackOnError` allows a server application to receive a notification when the sent response is in error.

2.3 Real-time FileAct features

This section lists the features related to the ability of the vendor messaging interface to support SwiftNet FileAct and file management functions.

2.3.1 File processing features (BSMS)

The table below lists the general features required by vendor messaging interfaces supporting real-time FileAct.

Feature	Note	M/O	Sup
Send files by sending a `PutFileRequest`	BSMS.1	M	Y
Receive files by receiving a `PutFileRequest`	BSMS.2	M	Y
Download files by sending a `GetFileRequest`	BSMS.3	O	Y
Provide a file by receiving a `GetFileRequest`	BSMS.4	O	Y
Send file with 4-eyes approval	BSMS.5	O	Y

Note

BSMS.1 The vendor interface is sending a `PutFileRequest`. The file is prepared by the business application and made available in some way. Support of 2GB files is mandatory.

BSMS.2 The vendor interface is receiving a `PutFileRequest`. Support of 2GB files is mandatory.

BSMS.3 The vendor messaging interface is downloading a file from a server by sending a `GetFileRequest`. Support of 2GB files is mandatory.

BSMS.4 The vendor messaging interface is acting as a file server and process a received `GetFileRequest`. Support of 2GB files is mandatory. Note that you cannot avoid receiving a `GetFileRequest`. If the feature is not supported, an error must be returned in the `GetFileResponse`.

BSMS.5 This is a step in the flow of sending files where another operator is involved to approve the file emission. This can be implemented by business applications.

2.3.2 Real-time delivery monitoring (DLNT)

These are the features supporting the monitoring of file deliveries to counterparties.

Feature	Note	M/O	Sup
Ask for and receive Delivery Notifications	DLNT.1	M	Y
Send Delivery Notifications when requested	DLNT.2	M	Y

Note

DLNT.1 The vendor messaging interface can request and receive a positive delivery notification.

DLNT.2 The vendor messaging interface can send a delivery notification when requested.

2.3.3 File monitoring (FEV)

The table below lists the general features required by vendor messaging interfaces supporting FileAct.

Feature	Note	M/O	Sup
Process file events	FEV.1	M	Y

Note

FEV.1 This feature is required for keeping track of the file transfer status. It can be implemented either by using the file event subscription offered by the communication interface or by the file status commands. The choice depends on the resilience requirements. For example, highly resilient systems must not rely on the file status commands.

2.3.4 Remote file (RFH)

The table below lists the general features required by vendor messaging interfaces supporting FileAct.

Feature	Note	M/O	Sup
Support of Remote File Handler	RFH.1	O	Y

Note

RFH.1 The file handler is responsible for managing the access to physical files (emission, reception) on a different remote system, which does not contain an SNL instance, enabling a file to be transferred through FileAct.

Legal Notices

Copyright

Swift ©2026. All rights reserved.

Restricted Distribution

Do not distribute this publication outside your organisation unless your subscription or order expressly grants you that right, in which case ensure you comply with any other applicable conditions.

Disclaimer

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 and binding version.

Trademarks

Swift is the trade name of S.W.I.F.T. SC. The following are registered trademarks of Swift: Swift, the Swift logo, 3SKey, Innotribe, MyStandards, Sibos, SWIFTNet, SWIFT Institute, the Standards Forum logo, SWIFT gpi with logo, the SWIFT gpi logo, and UETR. Other product, service, or company names in this publication are trade names, trademarks, or registered trademarks of their respective owners.