# ISO20022 Documentation

## Instructions

### HOW-TO Use our library

All Swift MX messages are identified by a code id and a name. The code id looks like `FIToFICstmrCdtTrf` and is located in the `.xsd` file describing the XML schema of the current message. The name of the message is more descriptive and looks like `FIToFICustomerCreditTransfer`. Both the name and the code id of the message are available in the Swift MX messages catalogue.

For every Swift MX message there is an equivalent class. The name of the class is identical to the name of the message. For example the name of the class for the message named `FIToFICustomerCreditTransfer` is `FIToFICustomerCreditTransfer`.

There are three steps the user must follow in order to build a new Swift MX message:

1. Initialize the class corresponding to the message.
2. Add data to the class.
3. Validate the message.

#### Initialize the class corresponding to the message

The initialization of the class is as simple as initializing any class in Java. For the example message we are using here (`CustomerCreditTransferInitiation`) the initialization would be

```
Message message = new CustomerCreditTransferInitiation();
```

The above command will initialize a class for this message named `CustomerCreditTransferInitiation` which is initially empty.

##### Parsing a file

```
Message message = new CustomerCreditTransferInitiation();
message.parseAndValidate(new File("/path/to/pain.001.001.03.xml"));
```

#### Add data to the class

The next step is to add data to the new message. In order to add some data to the message, user must know which element in the message tree he wants to add. The element he wants to add is identified by an XML path. The value of the element user wants to add may be a `String`, a `Boolean` or a complex type that is described in the SWIFT MX type catalog.

So to enter some data into the message, user must call the following method of the previously instantiated object

```
message.setElement(path, value);
```

Where the path argument is a `String` identifying the element to add and the value argument is an `Object`.

User can also work with the XSD defined classes that represent a tag. e.g. `GroupHeader32` for `GrpHdr` tag.

#### Validate the message

After building a Swift MX message using the appropriate class, user may want to validate this message. Of course validation is not mandatory but is the only way to prove that the message is correct. Validation is performed by calling the `validate()` method and internally is a two step process:

1. Validation against the schema of the message in order to ensure that the message is a well formed instance of it.
2. Validation against any Validation Rule as described for that message by the SEPA rules.

To perform those two levels of validation user must call the following method:

```
ValidationErrorList errorList = message.validate();
```

The `validate()` method performs two level validation on the message and returns an `ArrayList` containing the validation errors that may occur. Each error is contained in the `ArrayList` as a `ValidationError` object describing the error.

### Code samples

#### Working with the demo

Our demo distribution comes with support for parsing/building/validating `pacs.002` messages. You can use our demo and see how our library works. The same code (with the equivalent classes and methods) is used for the other messages too.

**Parse/Validate pacs.002 message**

```
Message messageObject = new FIToFIPaymentStatusReport();
try {
        String messageContent = "";
        List<String> lines = Files.readAllLines(FileSystems.getDefault().getPath("/path/to/pacs.002.001.03.
xml"), StandardCharsets.UTF_8);
        for (String line : lines) {
                messageContent += line + "\r\n";
        }

        messageObject.parseXML(messageContent);
        ValidationErrorList errorList = messageObject.validate();
        errorList.forEach(error -> System.out.println(error.toString()));
        System.out.println(errorList.size());
        System.out.println(messageObject.convertToXML());
} catch (Exception e) {
        e.printStackTrace();
}
```

**Build a pacs.002 message**

```
try {
        FIToFIPaymentStatusReport messageObject = new FIToFIPaymentStatusReport();

        GroupHeader53 groupHeader = new GroupHeader53();
        groupHeader.setMsgId("MESSAGEID");

        GregorianCalendar cal = new GregorianCalendar();
        cal.setTime(new Date());
        XMLGregorianCalendar xmlDate;
        xmlDate = DatatypeFactory.newInstance().newXMLGregorianCalendar(cal);

        groupHeader.setCreDtTm(xmlDate);

        messageObject.getMessage().setGrpHdr(groupHeader);

        OriginalGroupHeader1 ogh = new OriginalGroupHeader1();
        ogh.setGrpSts(TransactionGroupStatus3Code.ACCP);
        ogh.setOrgnlMsgId("ORIGINALMESSAGEID");
        ogh.setOrgnlMsgNmId("pacs.008.001");

        messageObject.getMessage().setOrgnlGrpInfAndSts(ogh);

        System.out.println(messageObject.convertToXML());

        ValidationErrorList errorList = messageObject.validate();
        errorList.forEach(error -> System.out.println(error.toString()));
} catch (Exception e) {
        e.printStackTrace();
}
```

**Build a pacs.002 message (setElement)**

```
try {
        FIToFIPaymentStatusReport messageObject = new FIToFIPaymentStatusReport();

        GregorianCalendar cal = new GregorianCalendar();
        cal.setTime(new Date());
        XMLGregorianCalendar xmlDate = DatatypeFactory.newInstance().newXMLGregorianCalendar(cal);

        messageObject.setElement("GrpHdr/MsgId", "MESSAGEID");
        messageObject.setElement("GrpHdr/CreDtTm", xmlDate);

        messageObject.setElement("OrgnlGrpInfAndSts/OrgnlMsgId", "ORIGINALMESSAGEID");
        messageObject.setElement("OrgnlGrpInfAndSts/OrgnlMsgNmId", "pacs.008.001");
        messageObject.setElement("OrgnlGrpInfAndSts/GrpSts", TransactionGroupStatus3Code.ACCP);

        System.out.println(messageObject.convertToXML());

        ValidationErrorList errorList = messageObject.validate();
        errorList.forEach(error -> System.out.println(error.toString()));
} catch (Exception e) {
        e.printStackTrace();
}
```

## Demo

The demo web application showcases some of the functionalities of our MX libraries.

Paste or build your own message or select one of the predefines. The demo application lets you:

- parse text to message objects
- validate messages and see validation errors
- generate a sample PDF report