目次

[第1章 はじめに 4](#_Toc508933347)

[(1) 目的 4](#_Toc508933348)

[(2) 対象読者 4](#_Toc508933349)

[(3) 本書の内容 4](#_Toc508933350)

[(4) 謝辞 5](#_Toc508933351)

[第2章 金流商流連携 6](#_Toc508933352)

[(1) 金流商流連携の処理の流れ 6](#_Toc508933353)

[(2) 振込処理（振込明細挿入） 6](#_Toc508933354)

[(3) 入金確認処理（振込明細抽出） 7](#_Toc508933355)

[① ISO20022 Pain、Camtとは 8](#_Toc508933356)

[② 金融EDI情報格納領域とは 8](#_Toc508933357)

[③ Base64エンコードとは 10](#_Toc508933358)

[④ MIMEヘッダーとは 10](#_Toc508933359)

[⑤ 国連CEFACT Remittance Adviceとは 12](#_Toc508933360)

[第3章 金流情報作成方法 14](#_Toc508933361)

[(1) 総合振込（ISO20022 Pain.001）の作り方 14](#_Toc508933362)

[(2) 総合振込結果明細（ISO20022 Pain.002）の読込み方 23](#_Toc508933363)

[(3) 入出金明細（ISO20022　Camt.052）の読込み方 23](#_Toc508933364)

[(4) 振込入金通知（ISO20022　Camt.054）の読込み方 24](#_Toc508933365)

[(5) 総合振込依頼制御情報（ISO20022　Bah.001）の作り方 26](#_Toc508933366)

[(6) 総合振込結果制御情報（ISO20022　Bah.001）の作り方 31](#_Toc508933367)

[① JAXBの使用例 35](#_Toc508933368)

[第4章 金融ＥＤＩ情報作成方法 39](#_Toc508933369)

[(1) 商流情報項目 39](#_Toc508933370)

[(2) 国連CEFACT 支払案内メッセージの作り方 41](#_Toc508933371)

[(3) 国連CEFACT 基礎データ記述方法 45](#_Toc508933372)

[① 金額データ（Amount）型記述方法 45](#_Toc508933373)

[② バイナリ―オブジェクト型記述方法 45](#_Toc508933374)

[③ コード（Code）型記述方法 46](#_Toc508933375)

[④ 日付・時刻データ型記述方法 47](#_Toc508933376)

[⑤ IDデータ（Identifier）型記述方法 50](#_Toc508933377)

[⑥ 真偽値（Indicator）型記述方法 51](#_Toc508933378)

[⑦ 単位付き数値（Measure）型記述方法 51](#_Toc508933379)

[⑧ 数値（Numeric）型記述方法 51](#_Toc508933380)

[⑨ 数量（Quantity）型記述方法 51](#_Toc508933381)

[⑩ テキスト（Text）型記述方法 51](#_Toc508933382)

[第5章 参考文献 52](#_Toc508933383)

[第6章 付録 53](#_Toc508933384)

[(4) XMLスキーマ 53](#_Toc508933385)

[① Pain.001のXMLスキーマ（入力チェック用） 53](#_Toc508933386)

[② Pain.001のXMLスキーマ（出力チェック用） 53](#_Toc508933387)

[③ Camt.052のXMLスキーマ（入力チェック用） 54](#_Toc508933388)

[④ Camt.052のXMLスキーマ（出力チェック用） 54](#_Toc508933389)

[⑤ Camt.054のXMLスキーマ（入力チェック用） 54](#_Toc508933390)

[⑥ Camt.054のXMLスキーマ（出力チェック用） 54](#_Toc508933391)

[⑦ Remittance AdviceのXMLスキーマ（出力チェック用） 54](#_Toc508933392)

[⑧ Remittance AdviceのXMLスキーマ（入力チェック用） 54](#_Toc508933393)

[(5) コード一覧 54](#_Toc508933394)

[① 業界区分 54](#_Toc508933395)

[② データ区分 55](#_Toc508933396)

[(6) メッセージ例 55](#_Toc508933397)

[① Pain.001の例 56](#_Toc508933398)

[② Camt.052の例 56](#_Toc508933399)

[③ Camt.054の例 56](#_Toc508933400)

[④ Remittance Adviceの例 56](#_Toc508933401)

[(7) Pain.001メッセージ作成例 56](#_Toc508933402)

[(8) Camt.052メッセージ読込例 **エラー! ブックマークが定義されていません。**](#_Toc508933403)

[(9) Camt.054メッセージ読込例 **エラー! ブックマークが定義されていません。**](#_Toc508933404)

　はじめに

## 目的

ISO 20022 は、XMLを主要なデータ記述言語とした金融通信メッセージの国際規格である。ISO 20022 は、標準化された金融メッセージの作成を一義的な目的とするが、金融業務分野で利用されている様々な通信メッセージに対しインターオペラビリティ（相互運用性）を実現することが容易になる可能性がある。

全銀EDIシステムはISO20022のXMLメッセージを採用することで、大きな商流情報などの金融EDI情報を添付可能とする拡張を行っている。

しかし、これまでは金融通信メッセージとしては固定長電文が使われていたため、ISO20022のXMLメッセージを扱えるシステムはほとんど無い。このため、全銀EDIシステムを利用するためには、ISO20022のXMLメッセージを扱う機能を追加することが急務であるが、金融系のシステム・アプリケーションの開発者の多くはISO20022の様な大きなXMLメッセージの処理は不慣れでもある。

本ガイドは、ISO20022のXMLメッセージの作成や読み取りの具体的な方法を解説し、システム開発者の一助とすることを目的とする。

## 対象読者

全銀EDIシステムなどの金融サービスに接続するシステム・ソフトウェアにおけるISO20022のXMLメッセージを作成する部分を開発するが、以下の様な技術者に本ガイドを読むことを推奨する。

* + ISO20022 Pain、Camtメッセージを熟知してない技術者
  + 商流情報向けの国連CEFACTのフォーマット（XML）を熟知していない技術者
  + XML技術を熟知していない技術者

本ガイドでは、読者諸氏がJavaプログラミング言語を既に理解していることを想定している。

## 本書の内容

**未定稿**

第2章で金流商流連携について、第3章で金流情報作成方法、第4章で金融ＥＤＩ情報作成方法について記述する。また関連資料としてXMLスキーマ、メッセージ例、プログラム例を付録に添付している。

## 謝辞

**未定稿**

金流商流連携

## 金流商流連携の処理の流れ

支払情報および入金情報に付加情報（拡張金融EDI）を付与し交換する金流商流連携処理により、効率的な消込処理などが可能になる。



図 1　金流商流連携の流れ

具体的には、①支払企業が支払関連データをXML形式で送る。②プラットフォーム内で支払関連データを変換して支払企業の銀行へ送る。③支払い企業の銀行は現状通り、固定長・20桁でデータのやり取りを行う。④納入企業の銀行が入金通知データを固定長・20桁で送る。⑤プラットフォーム内で入金通知データを変換して納入企業へ送る。⑥入金通知データを受信し、入金確認、消込などの処理を行う。

## 振込処理（振込明細挿入）

振込は送金方法の一種で、金融機関に開設された預貯金口座に宛てて、金銭を払い込むことである。多くの場合、企業自らが金融機関に解説した預貯金口座から、請求や納品した企業の預貯金口座に金銭を払い込む方法として用いられる。

振り込まれた金銭は入金として振り込まれた企業は知りうるが、どの請求や納品に対する振込かを特定する消込を行っている。この消込の作業効率向上に効果的な方法として、振込の基となる請求や納品かを特定できる情報を振込明細として付加することが考えられた。

金融EDIシステムを使った振込では、Pain.001.001.03というXMLメッセージを使って行えるため、大きな商流情報でも金融EDI情報として付加することが可能となり、数多くの請求を一度に払うまとめ払いでも振込明細を付加することが出来るように成る。

振込に必要である口座情報等共通な情報は、多くの企業は基本契約として情報交換し、マスターデータに格納している。このため必要に応じてマスターデータに格納されている口座情報等も利用し、振込明細とともに総合振込情報（Pain.001.001.03）を作成し、金融機関に振込を依頼する。

金融EDIシステムでは、振込の依頼に対して総合振込結果明細（Pain.002.001.03）が作成される。取引明細別処理結果により振込処理の正常またはエラーが判別でき、個別のエラー内容は識別表示および仕向け金融機関指示情報に記載される。

## 入金確認処理（振込明細抽出）

入金が行われると振込入金通知（Camt.054.001.02）を金融機関が作成する。また入金や出金の状況が入出金明細（Camt.052.001.02）として金融機関が作成する。

通知された振込入金通知や入出金明細の振込明細にある金融EDI情報から商流情報を抽出し、送付された請求書・請求明細や振込明細と突合し、消込を行う。

### ISO20022 Pain、Camtとは

Painは国際規格ISO20022として標準化された送金指図（総合振込）を行うためのXML電文仕様であり、Camtも振込入金通知や入出金取引明細といった銀行から顧客企業に預金口座情報を通知するための国際標準XML電文仕様である。

Painの項目と総合振込の項目との相互の対応付け（マッピング）を行い、Painと総合振込の相互変換を可能とする。これによりこれまで使っていた総合振込の項目を使ってPainの電文の作成や、Painの電文のどの項目を使って振込を行うかが明確になる。



表 1 Painと総合振込の相互変換表例（一部）

誰もが同じ相互変換表を使うことで、Painの使い方が同じになる。

なお、上記はPainの例であるが、Camt（振込入金通知、入出金明細に関するXML電文）についても同様に項目の対応付け（マッピング）が必要となる。

全銀EDIシステムにおける変換表は全銀協のホームページ（<https://www.zenginkyo.or.jp/abstract/efforts/smooth/xml/>）に掲載されているので参照されたい。

### 金融EDI情報格納領域とは

全銀EDIシステムでは、総合振込はPain.001.001.03、振込入金通知はCamt.054.001.02、入出金明細はCamt.052.001.02のフォーマットを採用している。それぞれ商流情報を入れる領域としてRemittance InformationのUnstructured項目が指定されている。



Unstructured項目は140文字を記述でき、また繰り返し記述することが可能である。しかし、XMLのタグを記述することは許されていない。このためXML形式で記述されることが多い大きな商流情報は、そのままUnstructured項目に入れることが出来ない。

そこでバイナリデータなどを英数字と３種の記号（+、/、=）だけの文字列情報としてコード化するBase64エンコードを用いることが推奨されている。更に140文字以内の制限を守るためBase64エンコードで通常用いられている76文字毎の改行を使い、複数のUnstructured項目を繰り返し使う方法と、Base64エンコードされていることが明らかとなるようMIMEヘッダーの付加も推奨されている。

金融EDI情報格納領域には消込時の突合に使う商流情報を入れることになるが、格納する商流情報項目およびその情報の記述形式が明確化されないと取り出すことが出来ない。情報項目については、経済産業省が中心となり消込の突合に有益と思われる４０項目がまず選定されている。情報の記述形式については様々な方法が検討されているが、商取引が多様なことなどが要因として標準化はされていない。本ガイドでは一つの方式案（国連CEFACT Remittance Adviceメッセージ）を参考とし、商流情報を金融EDI情報にマッピングする一方式を金融EDI情報作成方法として解説する。



### Base64エンコードとは

Base64は、データを64種類の印字可能な英数字のみを用いて、印字可能な文字しか扱うことの出来ない通信環境にてマルチバイト文字やバイナリデータを扱うためのエンコード方式である。MIMEによって規定されていて、7ビットのデータしか扱うことの出来ない電子メールにて広く利用されている。具体的には、A–Z, a–z, 0–9 までの62文字と、記号2つ (+, /)、さらにパディング（余った部分を詰める）のための記号として = が用いられる。

### MIMEヘッダーとは

Multipurpose Internet Mail Extension（MIME）は、US-ASCIIのテキストしか使用できないインターネットの電子メールでさまざまなフォーマット（書式）を扱えるようにする規格である。通常はMIME（マイム）と略される。

MIMEは内容の記載フォーマットと伴にコンテンツ内容を明らかにする補助情報を記載するMIMEヘッダーについても規格化している。様々な内容の記載フォーマットを規定しているが、ここではXMLデータを記載する場合のMIMEヘッダーの具体例を以下に示す。

MIMEヘッダー

MIME-Version: 1.0

Content-Type: text/xml

Content-Transfer-Encoding: base64

MIME-Versionは、MIMEヘッダーの最初に記載することが決められており、MIMEの規定の版情報を表す。しかし、現在のところ版情報としては1.0しかない。しかし、「MIME-Version: 1.0」という行が先頭にあることで、それ以降はMIMEの規定に従えば解析を行えることが期待できる。「MIME-Version: 1.0」で始められていないUnstructured項目は平文として解析することとなる。

Content-Typeは内容の記述方法を表し、本例は「XMLデータ」であることを示している。Content-Transfer-Encodingは内容のエンコード方法を表し、本例は「base64エンコード」が行われていることを示している。

Base64エンコードおよびMIMEヘッダーの例を以下に示す。







### 国連CEFACT Remittance Adviceとは

国連CEFACT Remittance Adviceは、UN/CEFACT Cross Industry Remittance Adviceで支払案内という支払内容を支払先に通知する目的のメッセージで、XMLデータとして提示されている。

XMLデータは開始タグや終了タグといった共通の記述規則は決まっているが、どの様なタグを用いているか、またタグ付された項目の意味が何かなどは個別に規定される。特にタグ名などデータの構造に関する規定は、XMLデータを解析し業務に用いるデータとするために重要で、XMLスキーマとして記述される。

この様に多様なXMLスキーマが作られていくため、個々のXMLデータがどのXMLスキーマに従って作られているかを機械的に判別できるよう名前（名前空間名）を付与している。XMLスキーマに付与されている名前空間名と同じ名前をXMLデータにも記載されていると、持っているXMLスキーマで解析できるかを判別することが可能となる。国連CEFACT Remittance Adviceに付与されている名前空間名を以下に示す。

名前空間

urn:un:unece:uncefact:data:standard:CrossIndustryRemittanceAdvice:11

urnは名前の生成ルールの一つを表し、unは国連、uneceは国連欧州経済委員会、uncefactは国連CEFACTを表す。国連欧州経済委員会は国連CEFACTの上位機関である。dataはデータフォーマット、即ちメッセージを表し、他にcode（コード）、identifier（識別子）が使われている。CrossIndeustryRemiitanceAdviceは支払案内、11は11版を表している。

金流情報作成方法

XML電文の作成や読み込には、DOM、SAX、StAX、JAXB、Casterなど様々な方法がある。本ガイドではJAVAプログラミング言語環境で共通に利用可能なJAXB(スキーマからデータにバインディングする技術)を用いてJAVAライブラリを生成し、生成されたJAVAライブラリを活用し効率的にソフトウェアを作る方法を説明する。JAXBによるJAVAライブラリの作成方法については、「JAXBの使用例」を参照されたい。

JAXBを使い生成したJAVAライブラリを使うとソフトウェアが冗長と見える場合もあるが、データバインディング技術を使ったソフトウェアでは、タグ名のタイプミスや終了タグの作成漏れなどの不注意なミスが生じない。作成出来るタグはデータ型的にコンパイル時に検出されるためすり抜け防止が図れ、品質の高いソフトを効率的に作成する利点がある。

ここでは、JAVAの開発環境JDKがWindows7上にインストールされていることを想定している。また例に記載している変数名は特に意味は無い。

## 総合振込（ISO20022 Pain.001）の作り方

基本の処理の流れは以下となる。

* + 1. xjcコマンドでPain.001.001.03のXMLスキーマから関数を生成する。
    2. 生成された関数を使い、総合振込情報をJAVAオブジェクトに変換する。
    3. JDKのBase64関数を使い、金融EDI情報をBase64エンコードする。
    4. Unstrucrured項目に該当するJAVAオブジェクトに格納する。
    5. JAXBのmarshallerを使い、XMLに変換する。

なお、金融EDI情報の作り方は、第4章を参照されたい。

次にJAVAのプログラムを記述するが、まず準備として総合振込情報をJAVAオブジェクトに変換するためのxjcコマンドで生成した関数やランタイムルーチンをインポートする。

import javax.xml.bind.JAXBContext;

import javax.xml.bind.Marshaller;

import iso.std.iso.\_20022.tech.xsd.pain\_001\_001.\*;

import java.io.File;

import java.io.StringWriter;

生成したXMLを検証するためのランタイムルーチン等をインポートする。

import javax.xml.bind.Unmarshaller;

import javax.xml.transform.stream.StreamSource;

import java.io.StringReader;

import javax.xml.XMLConstants;

import javax.xml.validation.Schema;

import javax.xml.validation.SchemaFactory;

import javax.xml.validation.Validator;

日時データを作成するためのランタイムルーチン等をインポートする。

import javax.xml.datatype.XMLGregorianCalendar;

import javax.xml.datatype.DatatypeFactory;

import java.text.DateFormat;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.GregorianCalendar;

public class Main15 {

public static void main(String[] args) {

まず、以下のXMLに当たるJAVAオブジェクト群を生成する。

<Document>

<CstmrCdtTrfInitn>

<GrpHdr>

<MsgId>msgid</MsgId>

</GrpHd>

…

</CstmrCdtTrfInitn>

<Document>

まず、<Document></Document> と <CstmrCdtTrfInitn></CstmrCdtTrfInitn>　に当たるJAVAオブジェクト（Documentオブジェクト、CstmrCdtTrfInitnオブジェクト）を生成し、Documentオブジェクトの中にCstmrCdtTrfInitnオブジェクトを挿入する。

GrpHdrオブジェクトを生成し、CstmrCdtTrfInitnオブジェクトの中に挿入する。更にGrpHdrオブジェクトのMsgIdフィールドに文字列”msgid”を挿入する。

Document doc = new Document();

CustomerCreditTransferInitiationV03 ccti

= new CustomerCreditTransferInitiationV03();

doc.setCstmrCdtTrfInitn(ccti);

GroupHeader32 gh = new GroupHeader32();

ccti.setGrpHdr(gh);

gh.setMsgId("msgid");

次に、以下のXMLに当たるJAVAオブジェクト群を生成する。

<Document>

<CstmrCdtTrfInitn>

<GrpHdr>

…

<CreDtTm>2018-03-02T20:11:10.000+09:00</CreDtTm>

<NbOfTxs>1</NbOfTxs>

<InitgPty>

<Nm>PIx</Nm>

</InitgPty>

</GrpHdr>

…

</CstmrCdtTrfInitn>

<Document>

CreDtTmオブジェクトは、XMLのDateTime型を表現するxmlGregCal型のオブジェクトとして以下の様に生成し、GrpHdrオブジェクトのCreDtTmフィールドに挿入する。

try {

XMLGregorianCalendar xmlGregCal

= DatatypeFactory.newInstance()

.newXMLGregorianCalendar(new GregorianCalendar());

gh.setCreDtTm(xmlGregCal);

xmlGregCal.setYear(2018);

xmlGregCal.setMonth(3);

xmlGregCal.setDay(2);

xmlGregCal.setTimezone(9);

xmlGregCal.setTime(20, 11, 10);

// System.out.println(xmlGregCal);

} catch (Exception e) {

e.printStackTrace();

}

次にNbOfTxsフィールドに1を挿入する。更にInitgPtyオブジェクトを生成し、GrpHdrオブジェクトのInitgPtyフィールドに挿入する。InitgPtyオブジェクトののNmフィールドには、文字列”Pix”を挿入する。

gh.setNbOfTxs("1");

PartyIdentification32 pi = new PartyIdentification32();

pi.setNm("PIx");

gh.setInitgPty(pi);

GrpHdrオブジェクトの次に、以下のXMLに当たるJAVAオブジェクト群を生成する。

<Document>

<CstmrCdtTrfInitn>

…

<PmtInf>

<PmtInfId>pmtinfid1</PmtInfId>

<PmtMtd>TRF</PmtMtd>

<NbOfTxs>1</NbOfTxs>

<ReqdExctnDt>2018-03-02+09:00</ReqdExctnDt>

<Dbtr>

<Nm>PIz1</Nm>

</Dbtr>

<DbtrAcct>

<Id>

<Othr>

<Id>0010000001</Id>

</Othr>

</Id>

</DbtrAcct>

<DbtrAgt>

<FinInstnId/>

</DbtrAgt>

…

</PmtInf>

</CstmrCdtTrfInitn>

<Document>

次のPmtInfオブジェクトは複数作成が可能な為、ListオブジェクトをgetPmtInfメッソッドで取り出し、そこに生成したPmtInfオブジェクトを追加する。

次にPmtInfオブジェクトのPmtInfIdフィールドに文字列を挿入する。”pmtinfid1”は例で特に意味は無い。更にPmtInfオブジェクトのPmtMtdフィールドには、列挙型PaymentMethod3Codeの定数TRFを挿入する。またPmtInfオブジェクトのNbOfTxsフィールドに1を挿入する。

PaymentInstructionInformation3 pii = new PaymentInstructionInformation3();

ccti.getPmtInf().add(pii);

pii.setPmtInfId("pmtinfid1");

pii.setPmtMtd(PaymentMethod3Code.TRF);

pii.setNbOfTxs("1");

ReqdExctnDtオブジェクトは、XMLのDateTime型を表現するxmlGregCal型のオブジェクトとして以下の様に文字列で記述した日時を変換し生成し、GrpHdrオブジェクトのReqdExctnDtフィールドに挿入する。

try {

DateFormat format = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");

Date date = format.parse("2018-03-02 20:11:10");

GregorianCalendar cal = new GregorianCalendar();

cal.setTime(date);

XMLGregorianCalendar xmlGregCal

= DatatypeFactory.newInstance().newXMLGregorianCalendar(cal);

pii.setReqdExctnDt(xmlGregCal);

} catch (Exception e) {

e.printStackTrace();

}

PmtInfオブジェクトのDbtrオブジェクトに依頼人の名前Nmを付ける。文字列”Piz1”に意味は無い。次にPmtInfオブジェクトのDbtrAcctオブジェクトのIdフィールドに振込依頼人コードを挿入する。文字列"0010000001"に意味は無い。

pi = new PartyIdentification32();

pi.setNm("PIz1");

pii.setDbtr(pi);

CashAccount16 ca = new CashAccount16();

pii.setDbtrAcct(ca);

AccountIdentification4Choice ai = new AccountIdentification4Choice();

ca.setId(ai);

GenericAccountIdentification1 gai = new GenericAccountIdentification1();

gai.setId("0010000001");

ai.setOthr(gai);

銀行番号等を作成する。（作成途上）

BranchAndFinancialInstitutionIdentification4 bfii

= new BranchAndFinancialInstitutionIdentification4();

FinancialInstitutionIdentification7 fii

= new FinancialInstitutionIdentification7();

bfii.setFinInstnId(fii);

pii.setDbtrAgt(bfii);

MIMEヘッダーを挿入する。金融EDI情報をbase64エンコードし、Unstructuredタグを付加する。（未作成）

XML化の準備を行う。

String crlf = System.getProperty("line.separator");

StringWriter stringWriter = new StringWriter();

まず、marshaller機能を使ってXML化し、文字列領域に格納する。なお、テスト的に標準出力に表示している。

try {

JAXBContext context = JAXBContext.newInstance(Document.class);

Marshaller marshaller = context.createMarshaller();

try {

marshaller.marshal(doc, stringWriter);

System.out.print(stringWriter.toString());

System.out.print(crlf);

} catch (Exception e) {

e.printStackTrace();

}

XML宣言タグを陽に作成する例を示す。

marshaller.setProperty(Marshaller.JAXB\_FRAGMENT, true);

// 出力対象によりXML宣言タグの扱いが変わることに注意

try {

System.out.print("<?xml version=\"1.0\">");

marshaller.marshal(doc, System.out);

System.out.print(crlf);

} catch (Exception e) {

e.printStackTrace();

}

XML宣言タグを陽に作成し、清書する例を示す。

marshaller.setProperty(Marshaller.JAXB\_FRAGMENT, true);

// 出力対象によりXML宣言タグの扱いが変わることに注意

marshaller.setProperty(Marshaller.JAXB\_FORMATTED\_OUTPUT, true);

try {

System.out.print("<?xml version=\"1.0\">");

marshaller.marshal(doc, System.out);

} catch (Exception e) {

e.printStackTrace();

}

Unarshallerを使って検証を行う例を示す。ここはXML作成であり、バグの基に成りやすい無用な項目作成は抑止するため、出力チェック用のXMLスキーマを用いる。

try {

SchemaFactory sFactory

= SchemaFactory.newInstance(

XMLConstants.W3C\_XML\_SCHEMA\_NS\_URI);

Schema schema

= sFactory.newSchema(new File("pain.001.001.03.xsd"));

Unmarshaller unmarshaller = context.createUnmarshaller();

unmarshaller.setSchema(schema);

unmarshaller.setEventHandler(new MyValidationEventHandler());

Document customer = (Document) unmarshaller.unmarshal(

new StreamSource(new StringReader(stringWriter.toString())));

} catch (Exception e) {

e.printStackTrace();

}

出力チェック用のXMLスキーマを用い、XMLドキュメントの検証用のValidatorを使って検証を行う例を示す。

try {

SchemaFactory sFactory

= SchemaFactory.newInstance(

XMLConstants.W3C\_XML\_SCHEMA\_NS\_URI);

Schema schema

= sFactory.newSchema(new File("pain.001.001.03.xsd"));

Validator validator = schema.newValidator();

validator.validate(new StreamSource(

new StringReader(stringWriter.toString())));

} catch (Exception e) {

e.printStackTrace();

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

## 総合振込結果明細（ISO20022 Pain.002）の読込み方

**未定稿**

スキーマからデータバインディング技術（JAXB）を用いてJAVAライブラリを生成してプログラムを作成する。

入力チェック用のXMLスキーマを用い、標準としては任意でもシステム上必要な項目を必須としたチェックを行うなど厳しめのチェックを行う。

## 入出金明細（ISO20022　Camt.052）の読込み方

**未定稿**

スキーマからデータバインディング技術（JAXB）を用いてJAVAライブラリを生成してプログラムを作成する。

メインプログラム例

XJCコマンドで生成された関数と伴に、JAXBのunmarshaller を使ってCamt.052のメッセージをJAVAオブジェクトに変換する。

金流情報はXJCコマンドで生成された関数で参照する。

金融EDI情報は、まずUnstrucrured項目に該当するJAVAオブジェクトからBase64エンコード化された文字列を取り出す。

JDKのBase64関数を使い、Base64デコードしてXML化した金融EDI情報を取り出す。

import java.io.File;

import javax.xml.bind.JAXBContext;

import javax.xml.bind.Unmarshaller;

import iso.std.iso.\_20022.tech.xsd.camt\_052\_001.\*;

public class Main2 {

public static void main(String[] args) {

Document doc = new Document();

File file = new File("....xml");

try {

JAXBContext context = JAXBContext.newInstance(Document.class);

Unmarshaller unmarshaller = context.createUnmarshaller();

doc = (Document) unmarshaller.unmarshal(file);

} catch (Exception e) {

e.printStackTrace();

}

System.out.println("[" + doc.getBkToCstmrAcctRpt()

// TODO

.getID().getValue() + "]");

}

}

入力チェック用のXMLスキーマを用い、標準としては任意でもシステム上必要な項目を必須としたチェックを行うなど厳しめのチェックを行う。

## 振込入金通知（ISO20022　Camt.054）の読込み方

**未定稿**

メインプログラム例

XJCコマンドで生成された関数と伴に、JAXBのunmarshaller を使ってCamt.054のメッセージをJAVAオブジェクトに変換する。

金流情報はXJCコマンドで生成された関数で参照する。

金融EDI情報は、まずUnstrucrured項目に該当するJAVAオブジェクトからBase64エンコード化された文字列を取り出す。

JDKのBase64関数を使い、Base64デコードしてXML化した金融EDI情報を取り出す。

入力チェック用のXMLスキーマを用い、標準としては任意でもシステム上必要な項目を必須としたチェックを行うなど厳しめのチェックを行う。

## 総合振込依頼制御情報（ISO20022　Bah.001）の作り方

まず準備としてビジネスヘッダーをJAVAオブジェクトに変換するためのxjcコマンドで生成した関数やランタイムルーチンをインポートする。

import javax.xml.bind.JAXBContext;

import javax.xml.bind.Marshaller;

import iso.std.iso.\_20022.tech.xsd.head\_001\_001.\*;

import java.io.File;

import java.io.StringWriter;

import java.io.StringReader;

import javax.xml.XMLConstants;

import javax.xml.validation.Schema;

import javax.xml.validation.SchemaFactory;

import javax.xml.validation.Validator;

import javax.xml.transform.stream.StreamSource;

import javax.xml.datatype.XMLGregorianCalendar;

import javax.xml.datatype.DatatypeFactory;

public class Main10 {

public static void main(String[] args) {

ヘッダーの差出（From）関連を作成する。

AppHdr hdr = new AppHdr();

Party9Choice fr = new Party9Choice();

hdr.setFr(fr);

PartyIdentification42 frOrgId = new PartyIdentification42();

fr.setOrgId(frOrgId);

Party10Choice frId = new Party10Choice();

frOrgId.setId(frId);

OrganisationIdentification7 frOId = new OrganisationIdentification7();

frId.setOrgId(frOId);

GenericOrganisationIdentification1 frGOI

= new GenericOrganisationIdentification1();

frOId.getOthr().add(frGOI);

frGOI.setId(new CCPSW()

.setCC("03000000019999").setPSW("abcd").Unparse());

ヘッダーの宛先（To）関連を作成する。残りの項目も作成する。

Party9Choice to = new Party9Choice();

hdr.setTo(to);

BranchAndFinancialInstitutionIdentification5 toFIId

= new BranchAndFinancialInstitutionIdentification5();

to.setFIId(toFIId);

FinancialInstitutionIdentification8 toFId

= new FinancialInstitutionIdentification8();

toFIId.setFinInstnId(toFId);

ClearingSystemMemberIdentification2 toCSMId

= new ClearingSystemMemberIdentification2();

toFId.setClrSysMmbId(toCSMId);

toCSMId.setMmbId("To Fin. ID");

hdr.setBizMsgIdr("xxx");

hdr.setMsgDefIdr("head.001.001.01");

try {

XMLGregorianCalendar xmlGregCal

= DatatypeFactory.newInstance()

.newXMLGregorianCalendar(2018, 3, 20, 0, 0, 0, 0, 0);

hdr.setCreDt(xmlGregCal);

} catch (Exception e) {

e.printStackTrace();

}

XML化を図り、XMLスキーマ検証も行う。

String crlf = System.getProperty("line.separator");

StringWriter stringWriter = new StringWriter();

try {

JAXBContext context = JAXBContext.newInstance(AppHdr.class);

Marshaller marshaller = context.createMarshaller();

marshaller.marshal(hdr, stringWriter);

try {

SchemaFactory sFactory

= SchemaFactory.newInstance(

XMLConstants.W3C\_XML\_SCHEMA\_NS\_URI);

Schema schema

= sFactory.newSchema(new File("head.001.001.01.xsd"));

Validator validator = schema.newValidator();

validator.validate(new StreamSource(

new StringReader(stringWriter.toString())));

} catch (Exception e) {

e.printStackTrace();

}

marshaller.setProperty(Marshaller.JAXB\_FRAGMENT, true);

marshaller.setProperty(Marshaller.JAXB\_FORMATTED\_OUTPUT, true);

System.out.print("<?xml version=\"1.0\">");

marshaller.marshal(hdr, System.out);

} catch (Exception e) {

e.printStackTrace();

}

}

}

センター確認コードとパスワードは組み合せて記述するためのクラスを作成している。

public class CCPSW {

protected String cc;

protected String psw;

CCPSW() {

}

CCPSW(String cc, String psw) {

this.cc = cc;

this.psw = psw;

}

public String Unparse() {

return this.cc + ":" + this.psw;

}

public CCPSW Parse(String v) {

// TODO

return this;

}

public CCPSW setCC(String v) {

this.cc = v;

return this;

}

public String getCC() {

return this.cc;

}

public CCPSW setPSW(String v) {

this.psw = v;

return this;

}

public String getPSW() {

return this.psw;

}

}

## 総合振込結果制御情報（ISO20022　Bah.001）の作り方

生成されたソースコードをカスタマイズして記述した例を示す。まずカスタマイズしたJAVAライブラリをhead\_001\_00aに置いてあると仮定する。

import javax.xml.bind.JAXBContext;

import javax.xml.bind.Marshaller;

import iso.std.iso.\_20022.tech.xsd.head\_001\_00a.\*;

import java.io.File;

import java.io.StringWriter;

import javax.xml.bind.Unmarshaller;

import java.io.StringReader;

import javax.xml.XMLConstants;

import javax.xml.validation.Schema;

import javax.xml.validation.SchemaFactory;

import javax.xml.validation.Validator;

import javax.xml.transform.stream.StreamSource;

import javax.xml.datatype.XMLGregorianCalendar;

import javax.xml.datatype.DatatypeFactory;

import java.text.DateFormat;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.GregorianCalendar;

public class Main11 {

public static void main(String[] args) {

連続して関数を呼び出せるので、記述を減らすことが出来る。下記の後半は短くしていない。但し、エラーが発生した場合の特定化が面倒になる場合がある。

AppHdr hdr = new AppHdr();

hdr.setBizMsgIdr("xxx");

hdr.setMsgDefIdr("head.001.001.01");

OrganisationIdentification7 frOId = new OrganisationIdentification7();

hdr.setFr(new Party9Choice()

.setOrgId(new PartyIdentification42()

.setId(new Party10Choice()

.setOrgId(frOId)

)));

frOId.getOthr()

.add(new GenericOrganisationIdentification1()

.setId(new CCPSW()

.setCC("03000000019999").setPSW("abcd").Unparse()

));

Party9Choice to = new Party9Choice();

hdr.setTo(to);

BranchAndFinancialInstitutionIdentification5 toFIId

= new BranchAndFinancialInstitutionIdentification5();

to.setFIId(toFIId);

FinancialInstitutionIdentification8 toFId

= new FinancialInstitutionIdentification8();

toFIId.setFinInstnId(toFId);

ClearingSystemMemberIdentification2 toCSMId

= new ClearingSystemMemberIdentification2();

toFId.setClrSysMmbId(toCSMId);

toCSMId.setMmbId("0001");

上記以降は同じであるため省略する。

以下にカスタマイズの例を示す。Xjcで生成した段階では、Set関数は返却値の無いvoid型である。それを自オブジェクトを返却する様に修正を行っている。

以下にカスタマイズ前の例を示す。

/\*\*

\* frプロパティの値を設定します。

\*

\* @param value

\* allowed object is

\* {@link Party9Choice }

\*

\*/

public void setFr(Party9Choice value) {

this.fr = value;

}

以下にカスタマイズ後の例を示す。

/\*\*

\* frプロパティの値を設定します。

\*

\* @param value

\* allowed object is

\* {@link Party9Choice }

\*

\*/

public AppHdr setFr(Party9Choice value) {

this.fr = value;

return this;

}

### JAXBの使用例

まずXMLスキーマからJAVAソースコードを生成するxjcコマンドを使いXMLスキーマから関数を生成する。

C:\tmp>xjc pain.001.001.03.xsd

スキーマの解析中...

スキーマのコンパイル中...

iso\std\iso\\_20022\tech\xsd\pain\_001\_001\AccountIdentification4Choice.java

iso\std\iso\\_20022\tech\xsd\pain\_001\_001\AccountSchemeName1Choice.java

…

iso\std\iso\\_20022\tech\xsd\pain\_001\_001\TaxRecordPeriod1Code.java

iso\std\iso\\_20022\tech\xsd\pain\_001\_001\package-info.java

名前空間から算出されたディレクトリに関数が生成される。生成されるディレクトリは名前空間名から自動的に計算される。Pain.001.001.03の場合以下の様になる。

名前空間名：urn:iso:std:iso:20022:tech:xsd:pain.001.001.03

ディレクトリ：iso\std\iso\\_20022\tech\pain\_001\_001

またJAVAのインポートで指定する場合は以下の様になる。個々のクラス名を直接指定してもよいが、XMLスキーマの変更によりクラス名が変更されることもある。

Import iso.std.iso.\_20022.tech.xsd.pain\_001\_001.\*;

次にｊavacコマンドでコンパイルする。

C:\tmp>dir iso\std\iso\\_20022\tech\xsd\pain\_001\_001

…

C:\tmp\iso\std\iso\\_20022\tech\xsd\pain\_001\_001 のディレクトリ

2018/03/07 17:52 <DIR> .

2018/03/07 17:52 <DIR> ..

2018/03/07 17:52 2,708 AccountIdentification4Choice.java

2018/03/07 17:52 2,563 AccountSchemeName1Choice.java

2018/03/07 17:52 2,546 ActiveOrHistoricCurrencyAndAmount.java

　…

C:\tmp>javac iso\std\iso\\_20022\tech\xsd\pain\_001\_001\\*.java

Pain.001.001.03から作成された関数の例を示す。

//

// このファイルは、JavaTM Architecture for XML Binding(JAXB) Reference Implementation、v2.2.8-b130911.1802によって生成されました

// <a href="http://java.sun.com/xml/jaxb">http://java.sun.com/xml/jaxb</a>を参照してください

// ソース・スキーマの再コンパイル時にこのファイルの変更は失われます。

// 生成日: 2018.03.02 時間 07:55:45 PM JST

//

package iso.std.iso.\_20022.tech.xsd.pain\_001\_001;

import java.util.ArrayList;

import java.util.List;

import javax.xml.bind.annotation.XmlAccessType;

import javax.xml.bind.annotation.XmlAccessorType;

import javax.xml.bind.annotation.XmlElement;

import javax.xml.bind.annotation.XmlType;

/\*\*

\* <p>CustomerCreditTransferInitiationV03 complex typeのJavaクラス。

\*

\* <p>次のスキーマ・フラグメントは、このクラス内に含まれる予期されるコンテンツを指定します。

\*

\* <pre>

\* &lt;complexType name="CustomerCreditTransferInitiationV03">

\* &lt;complexContent>

\* &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">

\* &lt;sequence>

\* &lt;element name="GrpHdr" type="{urn:iso:std:iso:20022:tech:xsd:pain.001.001.03}GroupHeader32"/>

\* &lt;element name="PmtInf" type="{urn:iso:std:iso:20022:tech:xsd:pain.001.001.03}PaymentInstructionInformation3" maxOccurs="unbounded"/>

\* &lt;/sequence>

\* &lt;/restriction>

\* &lt;/complexContent>

\* &lt;/complexType>

\* </pre>

\*

\*

\*/

作成された関数の例の続きを示す。

@XmlAccessorType(XmlAccessType.FIELD)

@XmlType(name = "CustomerCreditTransferInitiationV03", propOrder = {

"grpHdr",

"pmtInf"

})

public class CustomerCreditTransferInitiationV03 {

@XmlElement(name = "GrpHdr", required = true)

protected GroupHeader32 grpHdr;

@XmlElement(name = "PmtInf", required = true)

protected List<PaymentInstructionInformation3> pmtInf;

/\*\*

\* grpHdrプロパティの値を取得します。

\*

\* @return

\* possible object is

\* {@link GroupHeader32 }

\*

\*/

public GroupHeader32 getGrpHdr() {

return grpHdr;

}

/\*\*

\* grpHdrプロパティの値を設定します。

\*

\* @param value

\* allowed object is

\* {@link GroupHeader32 }

\*

\*/

public void setGrpHdr(GroupHeader32 value) {

this.grpHdr = value;

}

作成された関数の例の続きを示す。

/\*\*

\* Gets the value of the pmtInf property.

\*

\* <p>

\* This accessor method returns a reference to the live list,

\* not a snapshot. Therefore any modification you make to the

\* returned list will be present inside the JAXB object.

\* This is why there is not a <CODE>set</CODE> method for the pmtInf property.

\*

\* <p>

\* For example, to add a new item, do as follows:

\* <pre>

\* getPmtInf().add(newItem);

\* </pre>

\*

\*

\* <p>

\* Objects of the following type(s) are allowed in the list

\* {@link PaymentInstructionInformation3 }

\*

\*

\*/

public List<PaymentInstructionInformation3> getPmtInf() {

if (pmtInf == null) {

pmtInf = new ArrayList<PaymentInstructionInformation3>();

}

return this.pmtInf;

}

}

金融ＥＤＩ情報作成方法

情報項目については、経済産業省が中心となり消込の突合に有益と思われる４０項目がまず選定されている。しかし、情報の記述形式については様々な方法が検討されているが、商取引が多様なことなどが要因として標準化はされていない。

そのため、ここでは一つの方式案（国連CEFACT Remittance Adviceメッセージ）を参考とし、商流情報を金融EDI情報にマッピングする一方式を解説する。本作成方法を強く推奨するものでは無いが、作り方の参考にしていただきたいと思います。

## 商流情報項目

Unstructured項目の中に入れる商流情報は、経産省が主導して産業界が標準化を図り、以下の４０項目が選定されている。（付録１参照）

○管理上利用する項目

・業界区分（情報項目番号：UN01005486/UN01005472）

・データ区分（情報項目番号：UN01005481/UN01005472）

○最低限必要な項目

・支払通知番号（情報項目番号：UN01008372）（※1）

・支払通知発行日（情報項目番号：UN01008376）（※1）

・請求書番号（情報項目番号：UN01005580）（※2）

・支払人企業法人コード（情報項目番号：UN01008795/UN01005756/UN01005757）（※3）

（※1）支払対象債務・支払日・支払金額・支払方法（振込か電債か）を通知する文書に付すもの。該当する文書が存在しない場合は記載せず、金融機関側で自動付番（振込みの際に使われている既存の受付番号等を利用）。

（※2）請求書（ないしそれに類する書類）を発行していない場合は記載不要

（※3）法人マイナンバーを持たない事業者（個人事業主等）については記載不要

○IT化推進による事務合理化に必要と思われる項目

・受取人企業法人コード（情報項目番号：UN01008794/UN01005756/UN01005757）

・請求先企業名（情報項目番号：UN01008586/UN01005756/UN01005759）

・請求先企業法人コード（情報項目番号：UN01008586/UN01005756/UN01005757）

・支払金額（明細）（情報項目番号：UN01008478）

・税額（情報項目番号：UN01005833）

・税区分（情報項目番号：UN01005834）

・税率（情報項目番号：UN01005836）

○利用可能とすべき項目

・支払番号（情報項目番号：UN01008498）

・受取人企業連絡先電話番号（情報項目番号：UN01005860）

・支払人企業連絡先電話番号（情報項目番号：UN01005860）

・請求先連絡担当者（情報項目番号：UN01005720）

・請求先連絡先部門（情報項目番号：UN01005721）

・請求先電話番号（情報項目番号：UN01005860）

・行番号（情報項目番号：UN01008833/UN01008361/UN01008363）

・発注番号（情報項目番号：UN01005580）

・受注番号（情報項目番号：UN01005580）

・単価（情報項目番号：UN01005792）

・数量（情報項目番号：UN01011464）

・納入番号（情報項目番号：UN01005627）

・納入日時（情報項目番号：UN01005628）

・製品コード（情報項目番号：UN01005813）

・製品名（情報項目番号：UN01005815）

・支払内容（情報項目番号：UN01005560）

・契約名（情報項目番号：UN01005589）

・締日（情報項目番号：UN01012129）

・入金予定日（情報項目番号：UN01012130）

・納品伝票番号（情報項目番号：UN01008733/UN01008361/UN01008363）

・請求書発行日（情報項目番号：UN01005582）

・金額相殺理由コード（UN01011095/UN01011098)

・相殺金額（UN01011095/UN01011101)

・受取人企業名（情報項目番号：UN01008794/UN01005756/UN01005759）（※4）

・支払人企業名（情報項目番号：UN01008795/UN01005756/UN01005759）（※4）

・支払合計金額（情報項目番号：UN01008471）（※4）

・支払日時（情報項目番号：UN01008500）（※4）

（※4）XML電文移行対象取引（予定）に、下表に示すように既に代替可能と思われる項目が存在するため、EDI情報欄への記載不要との整理が可能と考えられる項目。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 今回整理案 | XML電文移行対象取引（代替候補案） | | |
| 総合振込 | 振込入金通知 | 入出金取引明細 |
| 項目名 | 受取人企業名 | 受取人名 | 口座名 | 口座名 |
| 支払人企業名 | 振込依頼人名 | 振込依頼人名 | 振込依頼人名 |
| 支払合計金額 | 振込金額 | 金額 | 取引金額 |
| 支払日時 | 取組日 | 勘定日、起算日 | 勘定日、預入・払出日 |

本プロジェクトにおいては、消込処理における突合項目をまず格納するとともに、請求書から選定された４０項目が抽出できるのであれば、必須項目に拘らず当該項目を格納することがよいと考える。突合項目を増やす際に既に格納している項目であれば、変更の影響が低減する。

商流情報は、注文、請求、支払の段階で作成される情報項目を引き継ぎ収集する。

## 国連CEFACT 支払案内メッセージの作り方

金融EDI情報は、商流情報から抽出された情報項目を国連CEFACT支払案内メッセージの形式に格納し、XML化を行う。

まず準備として国連CEFACT支払案内メッセージをJAVAオブジェクトに変換するためのxjcコマンドで生成した関数やランタイムルーチンをインポートする。

import javax.xml.bind.JAXBContext;

import javax.xml.bind.Marshaller;

import un.unece.uncefact.data.standard.crossindustryremittanceadvice.\_11.\*;

import java.io.File;

import java.io.StringWriter;

import javax.xml.bind.Unmarshaller;

import java.io.StringReader;

import javax.xml.XMLConstants;

import javax.xml.validation.Schema;

import javax.xml.validation.SchemaFactory;

import javax.xml.validation.Validator;

import javax.xml.transform.stream.StreamSource;

まず、以下のXMLに当たるJAVAオブジェクト群を生成する。

<CrossIndustryRemittanceAdvice

xmlns="urn:un:unece:uncefact:data:standard:CrossIndustryRemittanceAdvice:11">

<CIExchangedDocumentContext>

<BusinessProcessSpecifiedCIDocumentContextParameter>

<ID>01</ID>

</BusinessProcessSpecifiedCIDocumentContextParameter>

<SubsetSpecifiedCIDocumentContextParameter>

<ID>JPSFEDI</ID>

</SubsetSpecifiedCIDocumentContextParameter>

</CIExchangedDocumentContext>

<CIRExchangedDocument>

<ID></ID>

<IssueDateTime><DateTime>1970-01-01T01:01:01</DateTime></IssueDateTime>

<IncludedCINote>

<Content></Content>

</IncludedCINote>

</CIRExchangedDocument>

…

</CrossIndustryRemittanceAdvice>

最初のIDまでのオブジェクトを作成する。

CrossIndustryRemittanceAdvice cira

= new CrossIndustryRemittanceAdvice();

CIExchangedDocumentContextType cedc

= new CIExchangedDocumentContextType();

CIDocumentContextParameterType cdcp

= new CIDocumentContextParameterType();

IDType id = new IDType();

cira.setCIExchangedDocumentContext(cedc);

cedc.setBusinessProcessSpecifiedCIDocumentContextParameter(cdcp);

cdcp.setID(id);

id.setValue("01");

行番号LineIDを作成する。

CIRTSupplyChainTradeTransactionType csctt

= new CIRTSupplyChainTradeTransactionType();

CIRDocumentLineDocumentType cdld

= new CIRDocumentLineDocumentType();

IDType lid = new IDType();

cira.setCIRTSupplyChainTradeTransaction(csctt);

csctt.setAssociatedCIRDocumentLineDocument(cdld);

cdld.setLineID(lid);

lid.setValue("L01");

言語をjaとしてNoteを記述する。

CINoteType cn = new CINoteType();

TextType tt = new TextType();

cdld.getIncludedCINote().add(cn);

cn.setContent(tt);

tt.setLanguageID("ja");

tt.setValue("memo");

言語をjaとして２個目のNoteを記述する。

cn = new CINoteType();

tt = new TextType();

cdld.getIncludedCINote().add(cn);

cn.setContent(tt);

tt.setLanguageID("ja");

tt.setValue("xyz");

他の項目を作成する。（未作成）

XML化を図り、XMLスキーマ検証も行う。

String crlf = System.getProperty("line.separator");

StringWriter stringWriter = new StringWriter();

try {

JAXBContext context

= JAXBContext.newInstance(CrossIndustryRemittanceAdvice.class);

Marshaller marshaller = context.createMarshaller();

marshaller.marshal(cira, stringWriter);

try {

SchemaFactory sFactory

= SchemaFactory.newInstance(

XMLConstants.W3C\_XML\_SCHEMA\_NS\_URI);

Schema schema

= sFactory.newSchema(

new File("CrossIndustryRemittanceAdvice.xsd"));

Validator validator = schema.newValidator();

validator.validate(new StreamSource(

new StringReader(stringWriter.toString())));

} catch (Exception e) {

e.printStackTrace();

}

marshaller.setProperty(Marshaller.JAXB\_FRAGMENT, true);

// 出力対象によりXML宣言タグの扱いが変わることに注意

marshaller.setProperty(Marshaller.JAXB\_FORMATTED\_OUTPUT, true);

System.out.print("<?xml version=\"1.0\">");

marshaller.marshal(cira, System.out);

} catch (Exception e) {

e.printStackTrace();

}

}

}

## 国連CEFACT 基礎データ記述方法

### 金額データ（Amount）型記述方法

金額データ（Amount）型は金額を表現する数値を指定するために使用する。通貨単位を明示的または暗黙的に指定する。

金額データ型には、補足情報として通貨コードと通貨コードリストの版情報を付加することが出来る。通貨コードはXML上ではCurrencyID属性、通貨コードリスト版情報はCurrencyCodeListVersionID属性として記述する。通貨コードはISO4217あるいは国連CEFACT推奨（UN/ECE Recommendation）の9番で規定されている3文字コードを用いる。

|  |  |  |
| --- | --- | --- |
| 補足情報名 | ＸＭＬ属性名 | 値 |
| 通貨コード | CurrencyID | ISO4217あるいは国連CEFACT推奨の9番（UN/ECE Recommendation 9）で規定されている3文字コード |
| 通貨コードリストの版情報 | CurrencyCodeListVersionID |  |

金額データ型の値はXMLスキーマのdecimal型で記述する。

例えばタグ名をAmountとし、通貨コードは円（JPY）、通貨コードリスト版情報は指定しないとしたXML形式で100円を記述すると以下の様になる。

<Amount CurrencyID=”JPY”>100</Amount>

規定上は通貨コード、通貨コードリスト版情報は任意記述であるが、金額データ型の場合、誤解を防ぐために通貨コードの記述を推奨する。しかし、XMLスキーマのデフォルトまたは必須として指定されている場合は、記述が無くても明らかなため省略してもよい。

### バイナリ―オブジェクト型記述方法

バイナリ―オブジェクト型は画像等の仕様化されたデータを指定するために使用する。データ処理が可能な様にフォーマット等の仕様を明示的または暗黙的に指定する。

バイナリ―オブジェクト型には、補足情報としてフォーマット情報、MIMEコード、エンコーディングコード、文字コード、URL、ファイル名を付加することが出来る。

フォーマット情報はXML上ではFormat属性、MIMEコードはmimeCode属性、エンコーディングコードはencodingCode属性、文字コードはcharacterSetCode属性、URLはurl属性、ファイル名はfilename属性として記述する。

MIMEコードは、エンコーディングコード、文字コード、ファイル名はRFC2045、RFC2046、RFC2047に定められているコードや記述方法を用いる。

|  |  |  |
| --- | --- | --- |
| 補足情報名 | ＸＭＬ属性名 | 値 |
| フォーマット情報 | Format |  |
| MIMEコード | mimeCode | RFC2045（RFC2231、RFC6532）、RFC2046（RFC3676、RFC5174、RFC6657）、RFC2047（RFC2231）に定められているコードや記述方法を用いる。RFC2048（RFC6838、RFC4285）に定められている様にIANAが登録管理しているコードも参照されたい。 |
| エンコーディングコード | encodingCode |
| 文字コード | characterSetCode |
| ファイル名 | filename |
| URL | url |  |

バイナリ―オブジェクト型の値はXML上ではXMLスキーマのbase64binary型で記述する。Base64binary型はbase64エンコーディングされた文字列である。

例えばタグ名をBinaryObjectとし、MIMEコードをtext/xmlとし、エンコーディングコードを、フォーマット情報、文字コード、URL、ファイル名は指定しないとしたXML形式で値「abcdefghij」、即ちbase64エンコード結果を「YWJjZGVmZ2hpag== 」すると以下の様になる。

<BinaryObject mimeCode=”text/xml”>YWJjZGVmZ2hpag==</BinaryObject>

規定上はフォーマット情報、MIMEコード、エンコーディングコード、文字コード、URL、ファイル名は任意記述であるが、バイナリ―オブジェクト型の場合、誤解を防ぐためにMIMEコード、エンコーディングコードの記述を推奨する。しかし、XMLスキーマのデフォルトまたは必須として指定されている場合は、記述が無くても明らかなため省略してもよい。

### コード（Code）型記述方法

コード（Code）型は符号化された文字列を含む値に使用する。コード値を理解するためのコードリストの情報が明示的または暗黙的に指定する。

コード型には、補足情報としてコードリストの識別値と名前、コードリスト管理者の識別値、名前と版情報、コード名、言語識別子、URI、コードリスト体系URIを付加することが出来る。コードリストの識別値は、XML上ではlistID属性、コードリストの名前はlistName属性、コードリスト管理者の識別値はlistAgencyID属性、コードリスト管理者の名前はlistAgencyName属性、コードリスト管理者の版情報はlistVersionID属性、コード名はname属性、言語識別子はlanguageID属性、URIはlistURI属性、コードリスト体系URIはlistSchemeURI属性として記述する。

コードリスト管理者の識別値は原則としてUN/EDIFACTのデータエレメント3055（コード管理機関）コードリストで規定されているコードを用いる。コードリスト管理者の版情報は原則としてUN/EDIFACTのデータエレメント3055（コード管理機関）コードリストの版情報の値を用いるが、記述されていない場合は最新判とする。コードリストの識別子はURL等で指定することも出来るが、コードリスト管理者が管理するコードリストの識別子を用いることを推奨する。

|  |  |  |
| --- | --- | --- |
| 補足情報名 | ＸＭＬ属性名 | 値 |
| コードリストの識別値 | listID |  |
| コードリストの名前 | listName |  |
| コードリスト管理者の識別値 | listAgencyID | 原則としてUN/EDIFACTのデータエレメント3055（コード管理機関）コードリストで規定されているコードを用いる。 |
| コードリスト管理者の名前 | listAgencyName |  |
| コードリスト管理者の版情報 | listVersionID | 原則としてUN/EDIFACTのデータエレメント3055（コード管理機関）コードリストの版情報の値を用いるが、記述されていない場合は最新判とする。 |
| コード名 | name |  |
| 言語識別子 | languageID |  |
| URI | listURI |  |
| コードリスト体系URI | listSchemeURI |  |

コード値は補足情報で指定されるコードリストの値とする。コード型の値はXML上ではXMLスキーマのtoken型で記述する。原則としてコード値の値に制限されるため、XMLスキーマで検証することが出来る。

例えばタグ名をCodeとし、コードリストの識別値は3055、コードリスト管理者の識別値は6、コードリストの名前、コードリスト管理者の名前と版情報、コード名、URI、コードリスト体系URIは指定しないとしたXML形式でコード値402を記述すると以下の様になる。なお、コードリスト管理者の識別値は6はUN/ECE（国連CEFACT）を表し、コードリストの識別値3055は国連CEFACTが管轄するUN/EDIFACTのコード管理機関コードリストを表し、コード値402はコード管理機関コードリストに登録されている国税庁になる。

<Code listID=”3055” listAgencyID=”6”>402</Code>

規定上はコードリストの識別値、名前と版情報、コードリスト管理者の識別値と名前、コード名、URI、コードリスト体系URIは任意記述であるが、コードリストの識別値とコードリスト管理者の識別値の記述を推奨する。しかし、XMLスキーマのデフォルトまたは必須として指定されている場合は、記述が無くても明らかなため省略してもよい。

### 日付・時刻データ型記述方法

日付・時刻データ型は日付と時刻を含む値に使用する。日付だけ、時刻だけ、日付と時刻の両方の3種類の指定方法がある。その他にフォーマットを陽に指定して記述する方法がある。

日付だけを指定する際には形式 "YYYY-MM-DD" を用いる。YYYY は年を、MM は月を、DD は日を表し、全コンポーネントが必須である。

コード値は補足情報で指定されるコードリストの値とする。コード型の値はXMLスキーマのtoken型で記述する。原則としてコード値の値に制限され、XMLスキーマで検証することが出来る。

例えばタグ名をStartDateとし、2002年9月24日をXML形式で記述すると以下の様になる。Date

<StartDate><Date>2002-09-24</Date></StartDate>

タイムゾーンの指定には、次のように UTC 時間の日付の後ろに "Z" を追加して入力する。

<StartDate><Date>2002-09-24Z</Date></StartDate>

または、次のように UTC 日付の後ろにオフセットとして正または負の時間を追加して指定することができる。

<StartDate><Date>2002-09-24-06:00</Date></StartDate> または

<StartDate><Date>2002-09-24+06:00</Date></StartDate>

時刻だけを指定する際には形式 "hh:mm:ss" を用いる。hh は時間を、mm は分を、ss は秒を表し、全コンポーネントが必須である。

例えばタグ名をStartTimeとし、9時30分10秒をXML形式で記述すると以下の様になる。

<StartTime><Time>09:30:10</Time></StartTime>

例えば9時30分10.5秒の様な1秒未満の値がある時刻をXML形式で記述することも出来て、以下の様になる。

<StartTime><Time>09:30:10.5</Time></StartTime>

タイムゾーンの指定には、次のように UTC の場合は時刻の後ろに "Z" を追加する。

<StartTime><Time>09:30:10Z</Time></StartTime>

または、次のように時刻の後ろにオフセットとして正または負の時間を追加して指定することができる。

<StartTime><Time>09:30:10-06:00</Time></StartTime> または

<StartTime><Time>09:30:10.5+06:00</Time></StartTime>

日付と時刻を指定する際には、形式 "YYYY-MM-DDThh:mm:ss" を用いる。YYYY は年を表し、MM は月を表し、DD は日を表し、T は必須な時刻セクションの開始を表し、hh は時間を表し、mm は分を表し、ss は秒を表し、注:全コンポーネントが必須である。

例えばタグ名をStartとし、2002年9月24日9時30分10秒および2002年9月24日9時30分10.5秒をXML形式で記述すると以下の様になる。

<Start><Date>2002-09-24T09:30:10</Date></Start>

<Start><Date>2002-09-24T09:30:10.5</Date></Start>

タイムゾーンの指定には、次のように UTCの場合は日時の後ろに "Z" を追加する。

<Start><Date>2002-09-24T09:30:10Z</Date></Start>

または、次のように日時の後ろにオフセットとして正または負の時間を追加して指定する。

<Start><DateTime>2002-09 T09:30:10-24-06:00</DateTime></Start> または

<Start><DateTime>2002-09 T09:30:10.5-24+06:00</DateTime></Stare>

フォーマットを陽に指定して記述する方法は、期間データなどに用いる。

フォーマットは補足情報として指定し、XML上ではformat属性およびタグDateTimeStringを記述する。

値はXML上ではXMLスキーマのstring型で記述する。このためXMLスキーマでフォーマットの検証は行われない。

期間データ型は、時間間隔を指定するために使用する。時間間隔は、形式 "PnYnMnDTnHnMnS" で指定する。

P は期間を表し(必須)、nY は年数を表し、nM は月数を表し、nD は日数を表し、T は時刻セクションの開始を表し(時間、分、秒を指定する場合は必須)、nH は時間数を表し、nM は分数を表し、nS は秒数を表す。

例えばタグ名をDurationとし、期間5年間、期間 5年2か月と10日、期間 5年2か月10日と15時間、期間 15時間をXML形式で記述すると以下の様になる。

<Duration format="PnY"><DateTimeString>P5Y</DateTimeString></ Duration>

<Duration format="PnYnMnD">

<DateTimeString>P5Y2M10D</DateTimeString>

</Duration>

<Duration format="PnYnMnDTnH ">

<DateTimeString>P5Y2M10DT15H</DateTimeString>

</Duration>

<Duration format=" PTnH ">

<DateTimeString>PT15H</DateTimeString>

</Duration>

P の前にマイナスを付けると負の期間を指定できる。

例えばタグ名をDurationとし、10日前から今日までの期間をXML形式で記述すると以下の様になる。

<Duration format="PnD"><DateTimeString>-P10D</DateTimeString></ Duration>

### IDデータ（Identifier）型記述方法

IDコード（Identifier）型は実体に対して符号化された文字列を対応させる際に使用する。IDコード値を理解するためのIDコードリストの情報を明示的または暗黙的に指定する。

IDデータ型には、補足情報としてIDリストの識別値と名前、IDリスト管理者の識別値、名前と版情報、IDリストのURI、IDリスト体系URIを付加することが出来る。IDリストの識別値は、XML上ではschemeID属性、IDリストの名前はschemeName属性、IDリスト管理者の識別値はschemeAgencyID属性、IDリスト管理者の名前はschemeAgencyName属性、IDリスト管理者の版情報はschemeVersionID属性、IDリストのURIはschemeDataURI属性、IDリスト体系URIはSchemeURI属性として記述する。

IDリスト管理者の識別値は原則としてUN/EDIFACTのデータエレメント3055（コード管理機関）コードリストで規定されているコードを用いる。IDリスト管理者の版情報は原則としてUN/EDIFACTのデータエレメント3055（コード管理機関）コードリストの版情報の値を用いるが、記述されていない場合は最新判とする。IDリストの識別値はコードリスト管理者が管理するIDリストの識別値の一覧から指定することを推奨する。

|  |  |  |
| --- | --- | --- |
| 補足情報名 | ＸＭＬ属性名 | 値 |
| IDリストの識別値 | schemeID |  |
| IDリストの名前 | schemeName |  |
| IDリスト管理者の識別値 | schemeAgencyID | 原則としてUN/EDIFACTのデータエレメント3055（コード管理機関）コードリストで規定されているコードを用いる。 |
| IDリスト管理者の名前 | schemeAgencyName |  |
| IDリスト管理者の版情報 | schemeVersionID | 原則としてUN/EDIFACTのデータエレメント3055（コード管理機関）コードリストの版情報の値を用いるが、記述されていない場合は最新判とする。 |
| IDリストのURI | schemeDataURI |  |
| IDリスト体系のURI | schemeURI |  |

IDコード値は補足情報で指定されるIDリストの値とする。IDコード型の値はXML上ではXMLスキーマのtoken型で記述する。

例えばタグ名をIDとし、IDリストの識別値は3055、IDリスト管理者の識別値は6、IDリストの名前、IDリスト管理者の名前と版情報、コード名、URI、コードリスト体系URIは指定しないとしたXML形式でコード値402を記述すると以下の様になる。なお、コードリスト管理者の識別値は413は国連CEFACT日本委員会を表し、IDリストの識別値JEC0001は国連CEFACT日本委員会が管理している業界区分コード表を表し、IDコード値JPSFEDIは金融EDIになる。

<ID schemeID=”JEC0001” schemeAgencyID=”413”>JPSFEDI</ID>

規定上はIDリストの識別値と名前、IDリスト管理者の識別値、名前と版情報、IDリストのURI、IDリスト体系のURIは任意記述であるが、IDリストの識別値とIDリスト管理者の識別値の記述を推奨する。しかし、XMLスキーマのデフォルトまたは必須として指定されている場合は、記述が無くても明らかなため省略してもよい。

### 真偽値（Indicator）型記述方法

**未定稿**

### 単位付き数値（Measure）型記述方法

**未定稿**

### 数値（Numeric）型記述方法

**未定稿**

### 数量（Quantity）型記述方法

**未定稿**

### テキスト（Text）型記述方法

**未定稿**

参考文献

[1] 全国銀行協会 ： XML 形式 適用業務およびレコード・フォーマット，平成29年8月，  
<https://www.zenginkyo.or.jp/fileadmin/res/abstract/efforts/smooth/xml/XML_news290829.pdf>

[2] ISO 20022 - Registration Authority ：Pain.001.001.03 XML schema,   
<https://www.iso20022.org/documents/messages/1_0_version/pain/schemas/pain.001.001.03.zip>

[3] ISO 20022 - Registration Authority ：Camt.052.001.02 XML schema,　  
<https://www.iso20022.org/documents/messages/1_0_version/camt/schemas/camt.052.001.02.zip> )

[4] ISO 20022 - Registration Authority ：Camt.054.001.02 XML schema,　  
<https://www.iso20022.org/documents/messages/1_0_version/camt/schemas/camt.054.001.02.zip>

[5] ISO 20022 - Registration Authority：ISO 20022 Message Archive,   
 <https://www.iso20022.org/message_archive.page>

[6] 経済産業省 中小企業庁 ： 金融 EDI 情報として格納すべき商流情報の整理について，平成28年 12月22日，<http://www.chusho.meti.go.jp/koukai/kenkyukai/kinyuedi/2016/161222kinyuedi.pdf>

[7] サプライチェーン情報基盤研究会 ： ビジネスインフラ 業界横断EDI仕様 V4，平成29年8月6日，<http://www.caos-a.co.jp/SIPS/bizinfra/CI_Spec4.html>

付録

**未定稿**

## XMLスキーマ

**未定稿**

### Pain.001のXMLスキーマ（入力チェック用）

国際標準(Pain.001.001.03)で規定しているメッセージ仕様および、全国銀行協会が付加した利用条件に合った総合振込(Pain.001.001.03)XMLメッセージと成っているかを検証するために利用できる。

### Pain.001のXMLスキーマ（出力チェック用）

作成した総合振込(Pain.001.001.03)XMLメッセージが、全銀EDIシステムで規定しているメッセージ仕様と合っているかを検証するために利用できる。

### Pain.002のXMLスキーマ（入力チェック用）

### Pain.002のXMLスキーマ（出力チェック用）

### Bah.001のXMLスキーマ（入力チェック用）

### Bah.001のXMLスキーマ（出力チェック用）

### Camt.052のXMLスキーマ（入力チェック用）

### Camt.052のXMLスキーマ（出力チェック用）

### Camt.054のXMLスキーマ（入力チェック用）

### Camt.054のXMLスキーマ（出力チェック用）

### Remittance AdviceのXMLスキーマ（出力チェック用）

### Remittance AdviceのXMLスキーマ（入力チェック用）

## コード一覧

**未定稿**

### 業界区分

**未定稿**

|  |  |  |  |
| --- | --- | --- | --- |
| コード表ID | DOMAIN | コード表名称 | コード管理機関 |
| 管理機関ID | JP, SIPS | 管理機関名称 | サプライチェーン情報基盤研究会 |
| コード値 | 名称 | 説明 | |
| JPSBASE | 業界横断EDI基本 | 業界横断EDIの基本となるメッセージで、各業務領域メッセージ設計における参照モデルとして利用する。 | |
| JPSLGPC | 自治体消耗品購買 | 豊田市役所一般購買のEDI実証実験で定義。 | |
| JPSSMED | 中小企業共通EDI | 中小企業共通EDI標準であり、基本、製造業、商社購買およびプロジェクト取引を含む。 | |
| JPSFEDI | 金融EDI | 金流商流情報連携で使われる金融関連メッセージ。 | |
| JPSSCDL | スケジューリングSCM | 需要予測／納入指示の組み合わせによるジャストインタイム方式EDI。 | |
| JPDR001 |  | 石川を中心とした水産業界（鮮魚）における共通EDI連携 | |
| JPDR002 |  | 北海道の中小企業における次世代共通EDI連携 | |
| JPDR003 |  | 大阪を中心とした中小製造業におけるビジネス情報共通EDI連携 | |
| JPDR004 |  | 貿易手続に係る輸出業界の受発注EDI連携 | |
| JPDR005 |  | 日用品の卸・小売業界における共通EDI連携 | |
| JPDR006 |  | 豊田商工会議所における商工会議所モデル共通EDI連携 | |
| JPDR007 |  | 碧南商工会議所における中小企業共通EDI連携 | |
| JPDR008 |  | 中小流通業界におけるクラウド型共通EDI連携 | |
| JPDR009 |  | 自動車業界における共通EDI連携 | |
| JPDR010 |  | 多摩地域活性化のためのビジネス情報共通EDI連携 | |
| JPDR011 |  | 水インフラ業界における共通EDI連携 | |
| JPDR012 |  | 静岡発エンジニアリングチェーンにおける共通EDI連携 | |

### データ区分

**未定稿**

## メッセージ例

**未定稿**

- 商流情報がエンコードされて組込まれたもの

メッセージ例

* + ISO20022（Pain.001、Camt.052、Camt.054）

- 商流情報がエンコードされて組込まれたもの

* + 国連CEFACT（Remittance Advice）

- 金流情報に組み込まれる元となるもの

### Pain.001の例

### Camt.052の例

### Camt.054の例

### Remittance Adviceの例

- 金流情報に組み込まれる元となるもの

## メッセージ例

### Pain.001の例

**未定稿**

金流情報とPain.001メッセージを作成し、XMLを標準出力に出す動作検証済のプログラム

### Camt.052の例

**未定稿**

Camt.052メッセージを読み込み、金流情報と金融EDI情報を標準出力に出す動作検証済のプログラム

### Camt.054の例

**未定稿**

Camt.054メッセージを読み込み、金流情報と金融EDI情報を標準出力に出す動作検証済のプログラム

### Pain.002の例

### Bah.001の例

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | | | |
|  |  | 2018年3月xx日 | 第0.1版発行 |  |
|  |  |  | | |
| 発行者 |  |  | | |
|  | | | | |