HITACHI
Inspire the Next

uCosminexus Interschema
ユーザーズガイド

解説・手引・操作書

3020-3-J38-20

マニュアルの購入方法

このマニュアル，および関連するマニュアルをご購入の際は，巻末の「ソフトウェアマニュアルのサービス ご案内」をご参照ください。

■対象製品

●適用 OS：AIX 5L V5.1，AIX 5L V5.2，AIX 5L V5.3
P-1M2C-BA31 uCosminexus Interschema 03-01
●適用 OS：HP-UX 11.0，HP-UX 11i，HP-UX 11i V2 (PA-RISC)
P-1B2C-BA31 uCosminexus Interschema 03-01
●適用 OS：HP-UX 11i V2 (IPF)，HP-UX 11i V3 (IPF)
P-1J2C-BA31 uCosminexus Interschema 03-01
●適用 OS：Solaris 8，Solaris 9
P-9D2C-BA31 uCosminexus Interschema 03-01
●適用 OS：Windows 2000，Windows XP，Windows Server 2003
P-262C-BS34 uCosminexus Interschema Java ライセンス 03-01
P-262C-BT34 uCosminexus Interschema API ライセンス 03-00
P-262C-BU34 uCosminexus Interschema 海上貨物通関ライセンス 03-00
P-262C-BV34 uCosminexus Interschema EDIFACT ライセンス 03-00
P-262C-BW34 uCosminexus Interschema CII ライセンス 03-00
P-262C-BX34 uCosminexus Interschema XML ライセンス 03-00
P-262C-BY34 uCosminexus Interschema GUI ライセンス 03-00
P-262C-BZ34 uCosminexus Interschema 03-01
これらのプログラムプロダクトのほかにもこのマニュアルをご利用になれる場合があります。詳細は「リリースノート」でご確認ください。

■輸出時の注意

本製品を輸出される場合には，外国為替および外国貿易法ならびに米国の輸出管理関連法規などの規制をご確認の上，必要な手続きをお取りください。
なお，ご不明な場合は，弊社担当営業にお問い合わせください。

■商標類

AIX は，米国における米国 International Business Machines Corp. の登録商標です。
HP-UX は，米国 Hewlett-Packard Company のオペレーティングシステムの名称です。
IBM は，米国およびその他の国における International Business Machines Corporation の商標です。
Java 及びすべての Java 関連の商標及びロゴは，米国及びその他の国における米国 Sun Microsystems, Inc. の商標または登録商標です。
JEDICOS は，（財）流通システム開発センターの登録商標です。
Microsoft は，米国およびその他の国における米国 Microsoft Corp. の登録商標です。
Sea-NACCS は，通関情報処理センターの登録商標です。
Solaris は，米国 Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。
Sun，Sun Microsystems，Java は，米国 Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。
UNIX は，X/Open Company Limited が独占的にライセンスしている米国ならびに他の国における登録商標です。
Windows は，米国およびその他の国における米国 Microsoft Corp. の登録商標です。
Windows Server は，米国 Microsoft Corporation の米国及びその他の国における登録商標です。
プログラムプロダクト「P-9D2C-BA31」には，米国 Sun Microsystems, Inc. が著作権を有している部分が

含まれています。
プログラムプロダクト「P-9D2C-BA31」には，UNIX System Laboratories, Inc. が著作権を有している部分が含まれています。

■発行

2005 年 6 月（第 1 版）3020-3-J38
2008 年 7 月（第 3 版）3020-3-J38-20

■著作権

変更内容

変更内容（3020-3-J38-20） uCosminexus Interschema　03-01

追加・変更内容	変更箇所
ettrans コマンドに，実行時オプション「-SNCODE」を追加した。	5.4.1(3)(a)，付録 J.4(2)，付録 J.4(3)
演算子，関数の説明で使用するデータ型の注意事項を追加した。	6.7.1
次の関数を追加した。 • 条件：IFEXIST • XML 操作：IFEXISTXML	表 6-14
メッセージ番号に「78」を追加した。	表 9-2，表 D-2
HP-UX の場合の共用ライブラリを追加・変更した。	9.4.2
Option クラスに，OPT_SNCODE フィールドを追加した。	10.2.4(1)，10.2.4(2)(q)
Sea-NACCS システムの更改に伴い，システム更改後の Sea-NACCS フォーマットをサポートした。 また，Sea-NACCS フォーマットについての説明を追加した。	12.5，表 E-2，表 F-1，付録 J.1，付録 J.2，付録 J.4，付録 L
メッセージを追加した。 KBET0078T-E	付録 B.3
メッセージを変更した。 KBET0058T-E	付録 B.3
「属性」の「n 型」に，日時を表す場合の説明を追加した。	表 J-2
「その他」に，「XML 電文」を追加した。	表 J-2
Sea-NACCS EDI 電文長を最大 500,000 バイトに変更した。	付録 J.2
適用 OS に，HP-UX 11i V3 (IPF) を追加した。	―

単なる誤字・脱字などはお断りなく訂正しました。

変更内容（3020-3-J38-10） uCosminexus Interschema　03-00

追加・変更内容
FDL エディタのリストビューを変更した。
MDL エディタのフォーマットツリーリストを変更した。
FDL エディタに外部出力コマンドを追加した。
MDL エディタで一度に複数のドキュメントを編集できるようにした。
ドラッグ & ドロップでフォーマット挿入する方法を追加した。
MDL エディタに次のコマンドを追加した。 • マージコマンド • 変数コマンド • 外部出力コマンド • プロパティ表示コマンド

追加・変更内容
MDL エディタで追加・変更できる式及び値の説明を追加した。
ettrans コマンドに次の実行時オプションを追加した。 • -FDS • -RELVL • -XND
ettrans コマンドに次の戻り値を追加した。 • 0x01000100 • 0x04000008 • 0x04000010
ettrans コマンドの NE オプションに指定するモード値「3」を追加した。
WHILE 関数を使用して，出現回数決定式を定義できるようにした。
文字コードの説明として，次の内容を追加した。 • 文字コード一覧 • 文字種別 • 文字コード変換 • EBCDIC ／ EBCDIK のコード表
変換の処理についての説明を追加した。
演算子に「i_null1 == i_null2」及び「i_null1 != i_null2」を追加した。
次の関数を追加・変更した。 • 型変換 ・TOSTRING ・HEXSTRTONUM ・HEXSTRTOSTREAM ・STREAMTOHEXSTR • XML 操作 ・GETTAGNAME ・GETANYTEXT ・EXISTXML ・EXISTCOUNTXML ・ARRAYREALSIZEXML ・GETCOMPXML ・GETCOMPIFXML • 特殊 ・EXISTWHILE ・WHILE • その他 ・GETCOMP ・GETCOMPIF
リネームの規則を追加した。
次の情報を出力できるようにした。 • ツリー情報ファイル • フォーマット情報ファイル • 差分情報ファイル また，出力情報としてマップ式ファイルを 7 章から付録へ移動した。
マージツールのオプション指定時の注意事項を追加した。

追加・変更内容
マージツールの戻り値に「802」を追加した。
次の関数に戻り値を追加した。 • 0x01000100 追加した関数 ・ETtrans2Init • 0x04000008 追加した関数 ・ETtrans2CreateMdlInfo ・ETtrans2ReleaseMdlInfo ・ETtrans2UpdateMdlInfo ・ETtrans2ReleaseThreadContext ・ETtrans2UpdateThreadContext • 0x04000010 追加した関数 ・ETtrans2CreateMdlInfo ・ETtrans2UpdateMdlInfo
Option クラスに次のオプションを追加した。 • OPT_FDS • OPT_XND
一部の再帰構造及び混在内容構造を強制的に ANY 要素として取り込むことができるようになった。
混在内容構造の説明を追加した。
XML データの変換の説明を追加した。
メッセージを追加した。 KBET0068T-S，KBET0073T-S，KBET0026J-S
メッセージを変更した。 KBET0012T-W，KBET0033T-E，KBET0038T-E
互換 API に次の戻り値を追加した。 • 0x01000100 • 0x04000010

はじめに

このマニュアルは，次に示すプログラムプロダクトの使用方法を説明したものです。

- P-1B2C-BA31　uCosminexus Interschema　03-01
- P-1J2C-BA31　uCosminexus Interschema　03-01
- P-1M2C-BA31　uCosminexus Interschema　03-01
- P-262C-BS34　uCosminexus Interschema Java ライセンス　03-01
- P-262C-BT34　uCosminexus Interschema API ライセンス　03-00
- P-262C-BU34　uCosminexus Interschema 海上貨物通関ライセンス　03-00
- P-262C-BV34　uCosminexus Interschema EDIFACT ライセンス　03-00
- P-262C-BW34　uCosminexus Interschema CII ライセンス　03-00
- P-262C-BX34　uCosminexus Interschema XML ライセンス　03-00
- P-262C-BY34　uCosminexus Interschema GUI ライセンス　03-00
- P-262C-BZ34　uCosminexus Interschema　03-01
- P-9D2C-BA31　uCosminexus Interschema　03-01

■対象読者

このマニュアルは，uCosminexus Interschema を使用して，電子データを変換する方を対象としています。また，CII（産業情報化推進センター）シンタックスルール，UN/EDIFACT（行政，商業，運輸のための電子データ交換に関する国連規格），及び XML について理解されていることを前提としています。

■マニュアルの構成

このマニュアルは，次に示す章と付録から構成されています。

第１章　概要

Interschema の概要と特長について説明しています。

第２章　データ変換の流れ

電子データを変換するために必要な作業の手順について説明しています。

第３章　FDL エディタの操作

FDL エディタの操作の概要，具体例を使用した FDL エディタの操作方法について説明しています。

第４章　MDL エディタの操作

MDL エディタの操作の概要，具体例を使用した MDL エディタの操作方法について説明しています。

第５章　変換のコマンド

トランスレータを使用してデータを変換する場合のコマンドについて説明しています。

第６章　定義と変換の規則
Interschema の定義と変換の規則について説明しています。

第７章　ユティリティ
Interschema で提供しているユティリティプログラムについて説明しています。

第８章　インタフェース
Interschema で提供しているインタフェースの概要，及び注意事項について説明しています。

第 9 章　データ変換処理 API（C 言語）
C 言語の実行環境で使用するデータ変換処理 API について説明しています。

第 10 章　データ変換処理 API（Java 言語）
Java 言語の実行環境で使用するデータ変換処理 API について説明しています。

第 11 章　ユーザ組み込み関数
ユーザ組み込み関数，及びユーザ組み込み関数が呼び出す出口関数について説明しています。

第 12 章　フォーマット作成時及びデータ変換時の注意事項
フォーマットを作成したり，データを変換したりする場合の注意事項を規格ごとに説明しています。

付録 A　トラブルシューティング
トランスレータでエラーが発生したときの処理について説明しています。

付録 B　トランスレータのメッセージ
トランスレータのメッセージの形式，見方，及び内容について説明しています。

付録 C　トランスレータの機能差異と電子データ交換ツールからの移行上の注意
トランスレータの機能差異と電子データ交換ツールから移行する場合の注意について説明しています。

付録 D　Interschema バージョン 1 との互換 API
Interschema バージョン 1 との互換 API について説明しています。

付録 E　Interschema から出力される情報
Interschema から出力される情報について説明しています。

付録 F　ファイルのバージョンとトランスレータのサポート範囲
ファイルのバージョンとトランスレータのサポート範囲について説明しています。

付録 G　CII シンタックスルールのサポート範囲
CII シンタックスルールのサポート範囲について説明しています。

付録 H　UN/EDIFACT のサポート範囲
UN/EDIFACT のサポート範囲について説明しています。

付録 I　JEDICOS のサポート範囲

JEDICOS のサポート範囲について説明しています。

付録 J　Sea-NACCS のサポート範囲

Sea-NACCS のサポート範囲について説明しています。

付録 K　文字コード

文字コードについて説明しています。

付録 L　用語解説

このマニュアルで使用している用語の意味を説明しています。

■読書手順

このマニュアルは，次の手順でお読みいただくことをお勧めします。

（凡例）

：　必ず読む項目　　：　必要に応じて読む項目

■このマニュアルで使用する略語

このマニュアルで使用する略語を次に示します。

略　語	正式名称
API	Application Programming Interface
CII	Center for the Informatization of Industry
CSV	Comma Separated Value
DLL	Dynamic Linking Library
DTD	Document Type Definition
EDI	Electronic Data Interchange
EIAJ	Electronic Industries Association of Japan
EJB	Enterprise Java Beans
Java VM	Java Virtual Machine
JEDICOS	Japan EDI for Commerce Systems
JNI	Java Native Interface
OS	Operating System
Sea-NACCS	Sea-Nippon Automated Cargo Clearance System
TFD	Transfer Form Data
UN/EDIFACT	United Nations Rules for Electronic Data Interchange for Administration, Commerce and Transport
XML	Extensible Markup Language

■このマニュアルで使用する記号

このマニュアルで使用する記号を次のように定義します。

記号	記号の意味
［ ］	ウィンドウ，メニュー，ダイアログ，キーボードのキー，及びタブの名称や項目を示します。
［A］＋［B］	＋の前に示した［A］キーを押しながら，＋の後ろに示した［B］キーを押すことを示します。
↑　↑	この記号で囲まれている計算式の値が小数点を含む場合，小数点以下を切り上げた整数値にすることを示します。 （例）　↑ 126.48 ↑ =127

コマンドの説明で使用する記号を次のように定義します。

記号	記号の意味
［ ］	この記号で囲まれている項目は省略できます。
{ }	この記号で囲まれている項目を 1 組として指定します。項目が記号	で区切られている場合は，そのうちの一つを指定します。

記号	記号の意味
| ストローク	複数の項目に対し，項目間の区切りを示し，「又は」の意味を示します。
...	この記号がある直前の項目は繰り返し指定できます。

■このマニュアルでの表記

このマニュアルでは，製品名称を次のように表記しています。

製品名称	表記	
AIX 5L for POWER V5.1	AIX	ワークステーションのOS
AIX 5L for POWER V5.2		
AIX 5L for POWER V5.3		
HP-UX 11.0	HP-UX	
HP-UX 11i		
HP-UX 11i V2 (PA-RISC)		
HP-UX 11i V2 (IPF)		
HP-UX 11i V3 (IPF)		
Solaris 8	Solaris	
Solaris 9		
Java(TM)	Java	

製品名称	表記	
Microsoft(R) Windows(R) 2000 Server Operating System	Windows 2000	Windows
Microsoft(R) Windows(R) 2000 Advanced Server Operating System		
Microsoft(R) Windows(R) 2000 Datacenter Server Operating System		
Microsoft(R) Windows(R) 2000 Professional Operating System		
Microsoft(R) Windows(R) XP Home Edition Operating System	Windows XP	
Microsoft(R) Windows(R) XP Professional Operating System		
Microsoft(R) Windows Server(R) 2003, Standard Edition 日本語版	Windows Server 2003	
Microsoft(R) Windows Server(R) 2003, Enterprise Edition 日本語版		
Microsoft(R) Windows Server(R) 2003, Standard x64 Edition Operating System		
Microsoft(R) Windows Server(R) 2003, Enterprise x64 Edition Operating System		
Microsoft(R) Windows Server(R) 2003 R2, Standard Edition Operating System		
Microsoft(R) Windows Server(R) 2003 R2 Enterprise Edition Operating System		
Microsoft(R) Windows Server(R) 2003 R2, Standard x64 Edition Operating System		
Microsoft(R) Windows Server(R) 2003 R2, Enterprise x64 Edition Operating System		
Cosminexus Interschema	トランスレータ	Interschema
uCosminexus Interschema		
Cosminexus Interschema - Light		
uCosminexus Interschema - Light		
Cosminexus Interschema - Definer	FDL エディタ及び MDL エディタ	
uCosminexus Interschema - Definer		
uCosminexus Interschema		

■ Windows のフォルダの表記

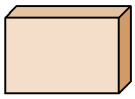
このマニュアルでは，Windows とワークステーションの OS で共通の操作の場合，Windows の「フォルダ」を「ディレクトリ」と表記しています。また，「¥」を「/」と表記しています。Windows の場合，「ディレクトリ」を「フォルダ」に，「/」を「¥」に置き換えてお読みください。

■図中で使用する記号

●入出力の動作

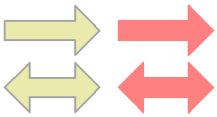
●ファイル又はデータベース

●プログラム

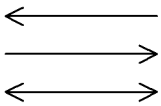


●ネットワーク

●データの流れ

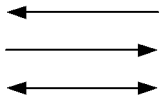


●制御の流れ

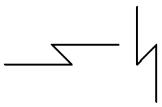


●工程，作業項目の流れ

●その他の流れ



●通信回線



■ヘルプの紹介

Windows の Interschema では，FDL エディタヘルプ，及び MDL エディタヘルプを提供しています。ヘルプでは，ダイアログ及びコマンドの詳細，属性及び文字コードの指定方法などを参照できます。

■ uCosminexus Interschema を使用する場合に必要なライセンスについて

Windows の Interschema では，FDL エディタ起動時，MDL エディタ起動時，及び変換実行時にライセンスが必要です。対応するライセンスを次に示します。

動作種別	規格種別	対応するライセンス
FDL エディタ起動時	－	uCosminexus Interschema GUI ライセンス
MDL エディタ起動時	－	
データ変換実行時	CII 又は CII3	uCosminexus Interschema CII ライセンス
	EDIFACT 又は EDIFACT4	uCosminexus Interschema EDIFACT ライセンス
	Sea-NACCS 又は Sea-NACCS 用 EDIFACT	uCosminexus Interschema 海上貨物通関ライセンス
	XML	uCosminexus Interschema XML ライセンス
データ変換処理 API（C 言語）使用時	－	uCosminexus Interschema API ライセンス

動作種別	規格種別	対応するライセンス
データ変換処理 API（Java 言語）使用時又は出口関数（Java）使用時	－	uCosminexus Interschema Java ライセンス

（凡例）

－：該当しません。

■常用漢字以外の漢字の使用について

このマニュアルでは，常用漢字を使用することを基本としていますが，次の用語については，常用漢字以外の漢字を使用しています。

溢れる（あふれる） 桁（けた） 汎用（はんよう） 必須（ひっす）

■ KB（キロバイト）などの単位表記について

1KB（キロバイト），1MB（メガバイト），1GB（ギガバイト），1TB（テラバイト）はそれぞれ 1,024 バイト，$1{,}024^2$ バイト，$1{,}024^3$ バイト，$1{,}024^4$ バイトです。

目次

図目次

表目次

1 概要

この章では，Interschema の概要と特長について説明します。

1.1 Interschema とは

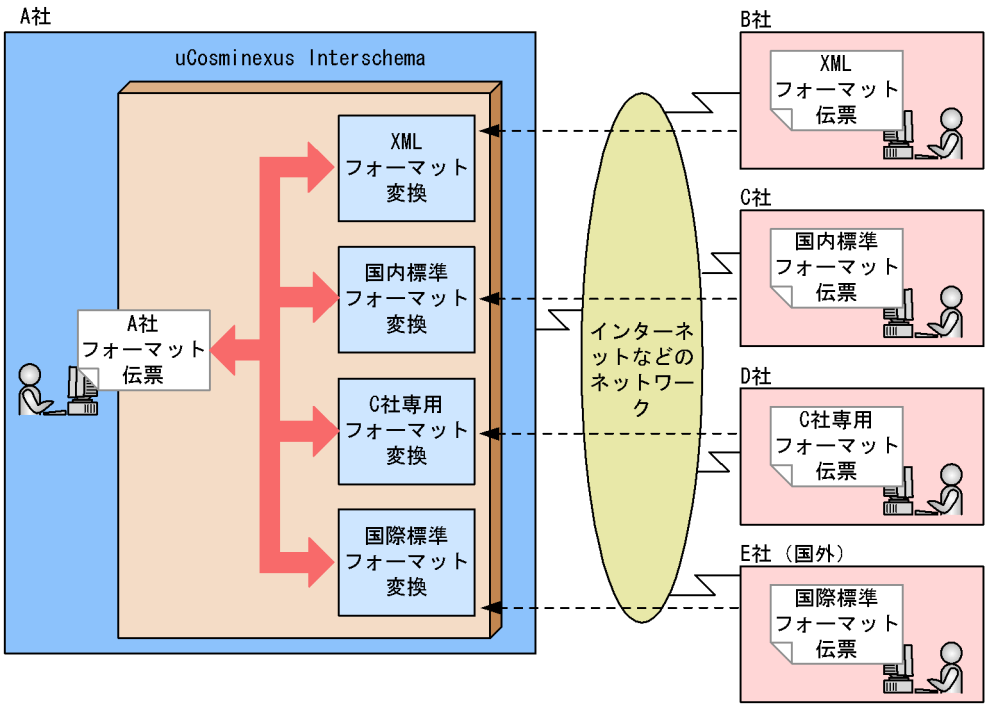
Interschema とは，自社が使用している伝票や帳票などのユーザデータのフォーマットを，他社でも使用できるように標準規格のデータフォーマットや他社固有のデータフォーマットに相互変換するためのトランスレータです。

1.1.1 Interschema の目的

Interschema は，EDI の国内標準及び国際標準の規格に対応しているため，国内だけでなく国外に及ぶ電子商取引も支援できます。

Interschema を利用した電子商取引の例を次に示します。

図 1-1 Interschema を利用した電子商取引の例

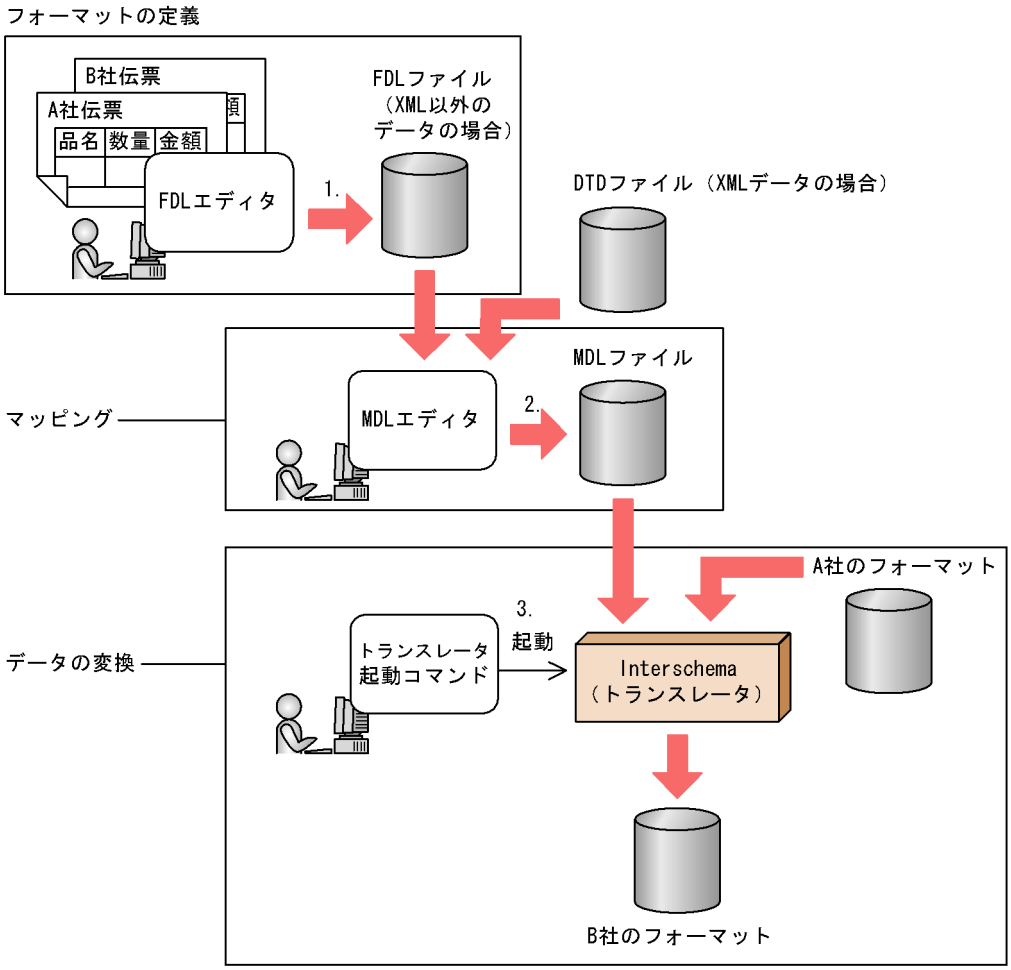


Interschema では，図 1-1 に示すように，ユーザ独自のフォーマットと，XML や EDI 標準規格フォーマット間のデータを相互に変換できます。また，自社や他社のユーザ独自のフォーマット間のデータも相互に変換できます。Interschema を使用することで，「データフォーマットの違いを意識しない電子商取引」を実現できます。

1.1.2　Interschema の機能

A 社フォーマットの伝票データを B 社フォーマットの伝票データに変換する場合を例にして，次に Interschema を使用したデータ変換の流れを示します。この例では，A 社及び B 社は，どちらもユーザ独自のフォーマットを使用しているものとします。

図 1-2　データ変換の流れ



1. **各社の伝票のフォーマットを定義する**
   各社の伝票のフォーマットを定義した FDL ファイルを作成します。FDL ファイルは，FDL **エディタ**で作成します。FDL エディタの操作方法については，「3.　FDL エディタの操作」を参照してください。

2. FDL **ファイル間を対応付ける**
   FDL ファイル間を対応付けた MDL ファイルを作成します。XML データを変換する場合は，FDL ファイルの代わりに DTD ファイルを用います。この FDL ファイル又

は DTD ファイル間を対応付ける操作を**マッピング**と呼びます。マッピングは，MDL **エディタ**で実行します。MDL エディタの操作方法については，「4. MDL エディタの操作」を参照してください。

3. **データを変換する**
   データを変換するときは，トランスレータの起動コマンドで MDL ファイル名を指定して，トランスレータを起動します。起動コマンドについては「5. 変換のコマンド」を参照してください。
   MDL ファイルで定義されている内容に従って，A 社のフォーマットが，B 社のフォーマットに変換されます。変換の規則については「6. 定義と変換の規則」を参照してください。

ここでは，ユーザ独自のフォーマット間のデータ変換について説明しましたが，例えば，B 社のデータのフォーマットが EDI 標準規格のフォーマットである場合にも同様にデータを変換できます。

また，データ変換を実現するために，Interschema は各種のユティリティ，API，及び関数を提供しています。ユティリティについては，「7. ユティリティ」を参照してください。

API，及び関数の概要については，「8. インタフェース」を参照してください。API の詳細については，「9. データ変換処理 API（C 言語）」，又は「10. データ変換処理 API（Java 言語）」を参照してください。関数の詳細については，「11. ユーザ組み込み関数」を参照してください。

1.2 Interschema の特長

Interschema の特長を次に示します。

（1） XML フォーマットのデータを変換できます

Interschema では，ユーザ独自のフォーマットのデータと XML フォーマットのデータを相互に変換できます。また，XML フォーマット間でデータを相互に変換できます。汎用性の高い XML フォーマットを適用することで，XML を解析するアプリケーションを利用し，データを再活用できます。

XML 文書を変換するための変換情報は，MDL エディタで，対応する DTD ファイルをフォーマットとして取り込んで作成します。

（2） ユーザフォーマットのデータを変換できます

Interschema では，ユーザ独自のフォーマット間でデータを相互に変換できます。

（3） EDI 標準規格フォーマットのデータを変換できます

Interschema では，ユーザ独自のフォーマットのデータと EDI 標準規格フォーマットのデータを相互に変換できます。また，EDI 標準規格のフォーマット間でデータを相互に変換できます。

EDI 標準規格のフォーマットでデータを変換する場合は，電子データ交換ツール又は Interschema が提供する EDI 標準規格の辞書のフォーマットを使用できます。
Interschema が対応している EDI 標準規格を次に示します。

- CII 標準
- EIAJ 標準
- UN/EDIFACT
- JEDICOS

また，海上貨物通関情報処理システム（Sea-NACCS）のフォーマットにも対応しています。

それぞれの内容について説明します。

(a) CII 標準

CII 標準は，財団法人 日本情報処理開発協会の産業情報化推進センターによって定められた国内での EDI の標準規格です。

(b) EIAJ 標準

EIAJ は，日本電子機械工業会によって定められた EDI 標準規格です。CII シンタックスルールの範囲内で，EIAJ 標準のデータに変換できます。

(c) UN/EDIFACT

UN/EDIFACT（行政，商業，運輸のための電子データ交換に関する国連規格）は，国際連合によって定められた国際的な EDI の標準規格です。

(d) JEDICOS

JEDICOS は，財団法人 流通システム開発センターによって定められた国内流通業界での EDI の標準規格です。EDI シンタックスルールは EDIFACT に従います。

(e) Sea-NACCS

Sea-NACCS は，厚生労働省や財務省などの各省庁と輸出入関連の民間業界をネットワークで接続し，通関に関する一連の手続きを処理するシステムです。このシステムで扱う電文には次の 2 種類があります。

- Sea-NACCS 形式の EDI 電文（Sea-NACCS EDI 電文）
- EDIFACT 形式の電文（Sea-NACCS EDIFACT 電文）

(4) フォーマットの構造をツリー構造で定義できます

Interschema では，伝票や帳票などのデータのフォーマットを FDL エディタで定義します。FDL エディタでは，データ構造をツリー構造で表現します。ツリー構造でデータをドラッグ&ドロップしたり，表示されるダイアログに定義内容を入力したりして構造を定義するため，複雑になりがちなデータ構造が分かりやすくなります。

フォーマットの定義例を，次に示します。

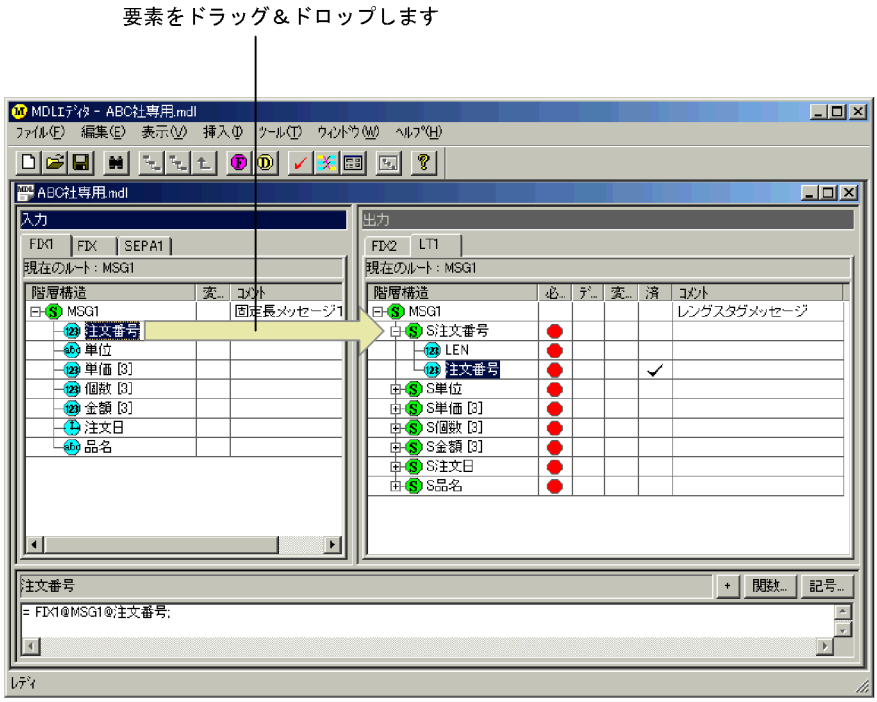
図 1-3 フォーマットの定義例

(5) フォーマットのマッピングはドラッグ & ドロップで定義できます

変換元のデータのフォーマットと変換後のデータのフォーマットとのマッピングは，MDL エディタで定義します。MDL エディタでは，画面上で要素をドラッグ＆ドロップして容易にマッピングできます。

データ要素のマッピングの例を，次に示します。

図 1-4　データ要素のマッピングの例



(6) 既存ユーザ業務にデータ変換処理を組み込めます

Interschema では，データ変換処理の API を提供しているので，既存のユーザ業務プログラムにデータ変換処理を組み込むことができ，ユーザ業務プログラムの機能を向上できます。なお，API を使用すると，ファイル化されているデータだけでなく，メモリ上にロードされているデータも変換できます。

(7) 変換情報の定義時にユーザ作成の関数を使用できます

変換情報（マッピング）の定義は式で記述できます。式には演算子や関数を使用できます。Interschema では，標準で提供している関数以外に，ユーザが定義した関数を組み込んで使用できるため，より複雑なデータ変換処理が実行できます。

2 データ変換の流れ

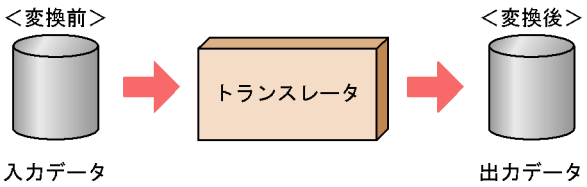
この章では，電子データを変換するために必要な作業の手順について説明します。

2.1　フォーマットの定義

この節では，具体的な例を使用して，データのフォーマット定義について説明します。フォーマットは FDL エディタで定義します。

2.1.1　フォーマットの種類

伝票や帳票のデータを変換するには，変換するデータの構造に基づいてフォーマットを定義します。Interschema では，トランスレータで変換する前の変換元のデータを入力データ，変換後のデータを出力データと呼びます。



データを変換するためには，入力データ及び出力データのそれぞれのフォーマットを定義する必要があります。

FDL エディタでは，「入力専用」「出力専用」「入出力兼用」という 3 種類のフォーマットを作成できます。各フォーマットの用途について説明します。

- 入力専用
  入力データ専用のフォーマットです。
- 出力専用
  出力データ専用のフォーマットです。
- 入出力兼用
  入力データと出力データどちらにも使用できるフォーマットです。

入力専用のフォーマット，又は出力専用のフォーマットは，データ構造の定義が簡単になります。また，入出力兼用フォーマットは入力データと出力データの構造にほとんど差がないような場合に使用すると便利です。作成するフォーマットの種類は，データの特性によって選択します。

2.1.2　フォーマットの定義内容

フォーマットに定義する内容は「型」「セパレータ」「構造」です。各内容について次に説明します。

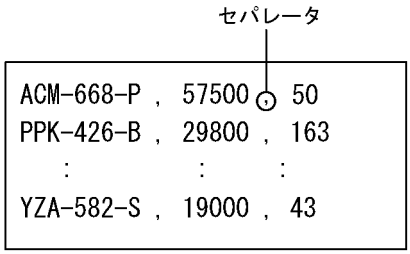
● 型

型とは，「製品名」や「製品コード」などのデータ中の項目となる要素の属性や要素の受け取る値などを定義したものです。

製品名	製品コード	単価(円)	個数	属性
部品□○	ABC-123-E	56,800	50	
部品×△	DEF-456-J	19,800	163	値
:	:	:	:	
部品○×	XYZ-789-S	3,900	2	

● セパレータ

セパレータとは，データを区切る記号です。セパレータには，コンマや改行などを使用します。

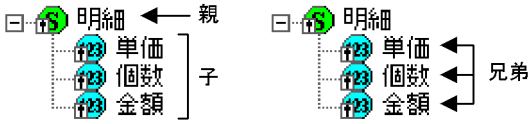


● 構造

構造とは，データの構成を表すもので，データの並び順になります。ツリー構造で階層的に表します。

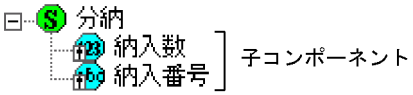
コンポーネント

構造の構成要素をコンポーネントと呼び，コンポーネントを組み合わせて構造を作成します。階層の異なるコンポーネント間は親子関係に，階層の同じコンポーネント間は兄弟関係にあります。



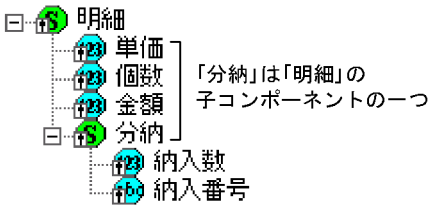
型のコンポーネント

コンポーネントには，型と構造があります。型のコンポーネントとは，型を定義した要素を構造のコンポーネントとして使用したもので，構造の子コンポーネントになります。例えば，構造「分納」は，型を定義した要素「納入数」「納入番号」という子コンポーネントで構成されています。



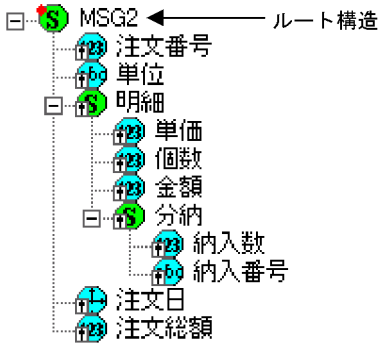
構造のコンポーネント

構造のコンポーネントとは，子コンポーネントを持ち，その構造自体がコンポーネントとなっているものです。例えば，構造「分納」は，「納入数」「納入番号」という子コンポーネントで構成される構造ですが，構造「明細」の子コンポーネントでもあります。



ルート構造

構造にはデータの最上位で，フォーマット内に一つだけあるルート構造があります。

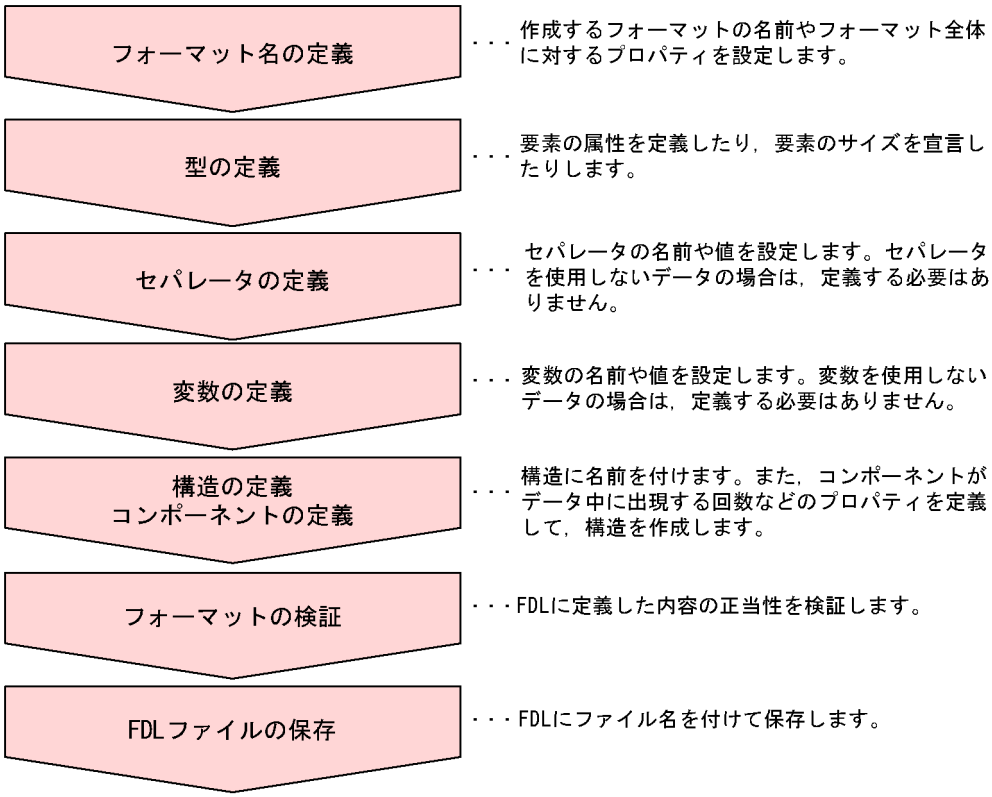


上の図では，構造「MSG2」がルート構造になります。データの順序と一致するように，コンポーネントを組み合わせてルート構造の下にデータの構造を定義します。

2.1.3 フォーマットの定義手順

FDL エディタでのフォーマット定義の手順を次に示します。

図 2-1　フォーマット定義の手順



例を使用して，フォーマットの定義について説明します。なお，以下のフォーマット定義例は，フォーマット定義の流れを理解しやすくすることを意図したものなので，フォーマット定義の詳細説明は省略しています。フォーマット定義の詳細については，「3.　FDL エディタの操作」を参照してください。

この例では，ユーザ独自のフォーマット帳票「IN」のデータを，ユーザ独自のフォーマット帳票「OUT」のデータに変換します。入力データが帳票「IN」のデータ，出力データが帳票「OUT」のデータとなるので，まず，帳票「IN」と帳票「OUT」のデータのフォーマットを定義します。

(1) 帳票「IN」のフォーマット

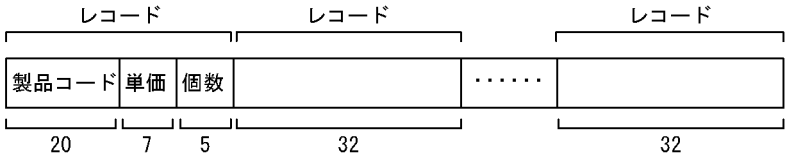
帳票「IN」を次に示します。

図 2-2 帳票「IN」

帳票「IN」のイメージ

	製品コード	単価(円)	個数
1	ABC-123-E	56,800	50
2	DEF-456-J	19,800	163
:	:	:	:
N	XYZ-789-S	3,900	2

帳票「IN」のデータフォーマット



（凡例） 数値はバイト数です。

帳票「IN」のレコードは,「製品コード」「単価」「個数」の三つの要素で構成されています。三つの要素はすべて固定長です。レコードの個数は不定です。

帳票「IN」のフォーマットを定義します。

(a) フォーマット名の定義

フォーマットを新規作成します。フォーマット挿入コマンドを実行して,［フォーマットのプロパティ］ダイアログでフォーマット名と，フォーマット全体に対するプロパティを設定します。

フォーマットが作成されます。

次に，フォーマットに要素の型を定義します。

(b) 型の定義

帳票「IN」には，「製品コード」「単価」「個数」という三つの要素があります。型挿入コマンドで，フォーマットに型を挿入して，型を定義します。

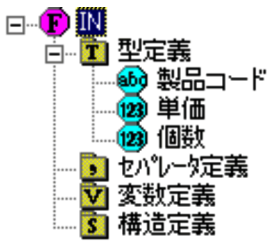
「製品コード」の型を定義します。［型のプロパティ］ダイアログで要素の名前，属性や種別を定義します。

［型のプロパティ］ダイアログで［詳細］ボタンをクリックします。表示された［型の詳細］ダイアログで要素の使用する文字コードやサイズなどを定義します。

型の詳細定義が済んだら，ダイアログを閉じます。

定義した要素「製品コード」の型が作成されます。

ほかの要素「単価」「個数」も同様に定義します。



(c) セパレータの定義

帳票「IN」ではセパレータを使用しないので，ここでは定義しません。

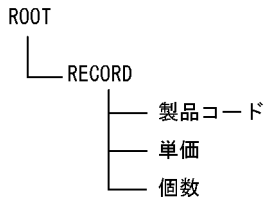
(d) 変数の定義

帳票「IN」では変数を使用しないので，ここでは定義しません。

次に，構造を定義します。

(e) 構造の定義

帳票「IN」のデータには，三つの要素で構成されるレコードが複数個あります。データの構造をツリーで表すと次のようになります。

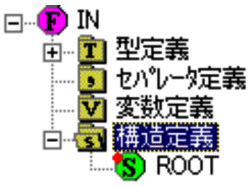


レコードを表す構造「RECORD」の下に，型を定義した三つの要素を型コンポーネントとして配置します。「RECORD」はルート構造「ROOT」の子コンポーネントとします。

まず，ルート構造「ROOT」を定義します。構造の挿入コマンドを実行して，フォーマットに構造を挿入します。

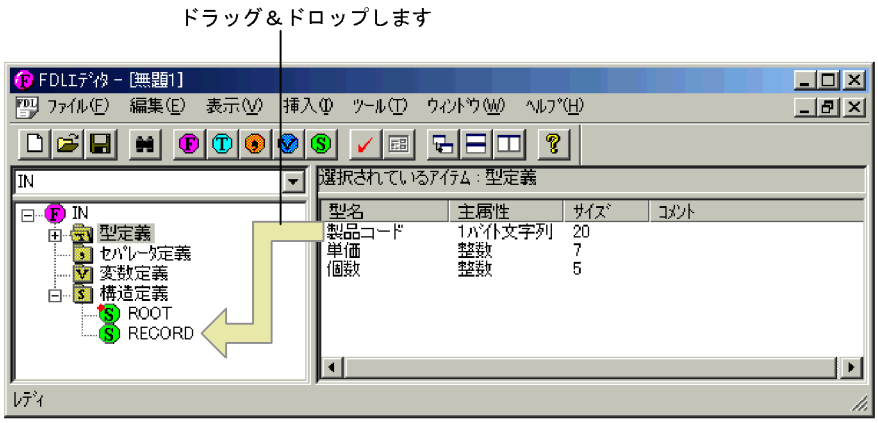
プロパティを定義したらダイアログを閉じます。

構造「ROOT」が作成されます。

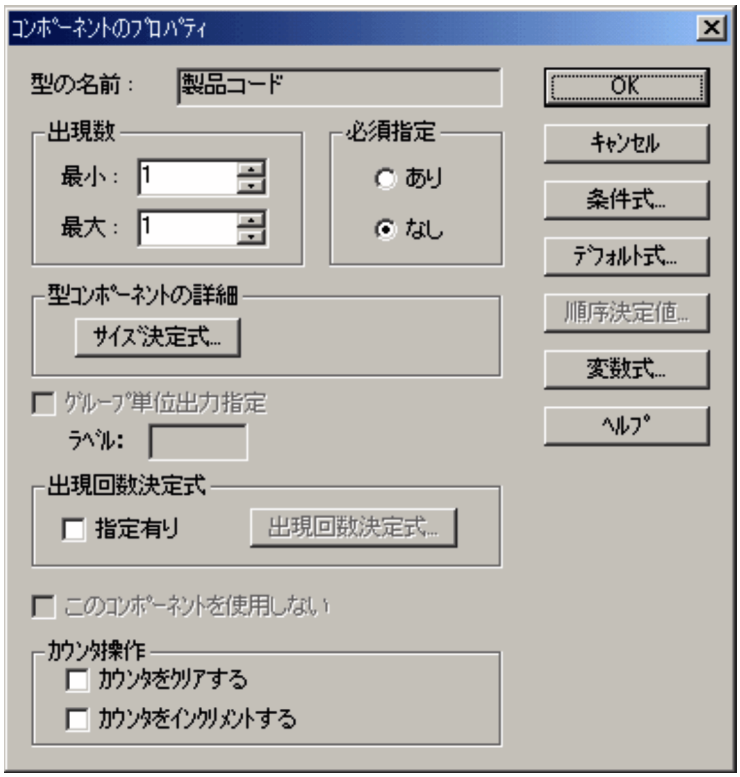


同様に構造「RECORD」を作成します。

構造「RECORD」の下に型を定義した三つの要素を配置します。要素の型を構造「RECORD」にドラッグ＆ドロップします。



［コンポーネントのプロパティ］ダイアログが表示されます。コンポーネントの出現回数などのプロパティを定義します。

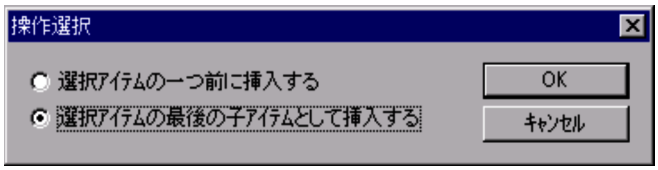


プロパティを定義したらダイアログを閉じます。

「製品コード」が構造「RECORD」のコンポーネントとして配置されます。

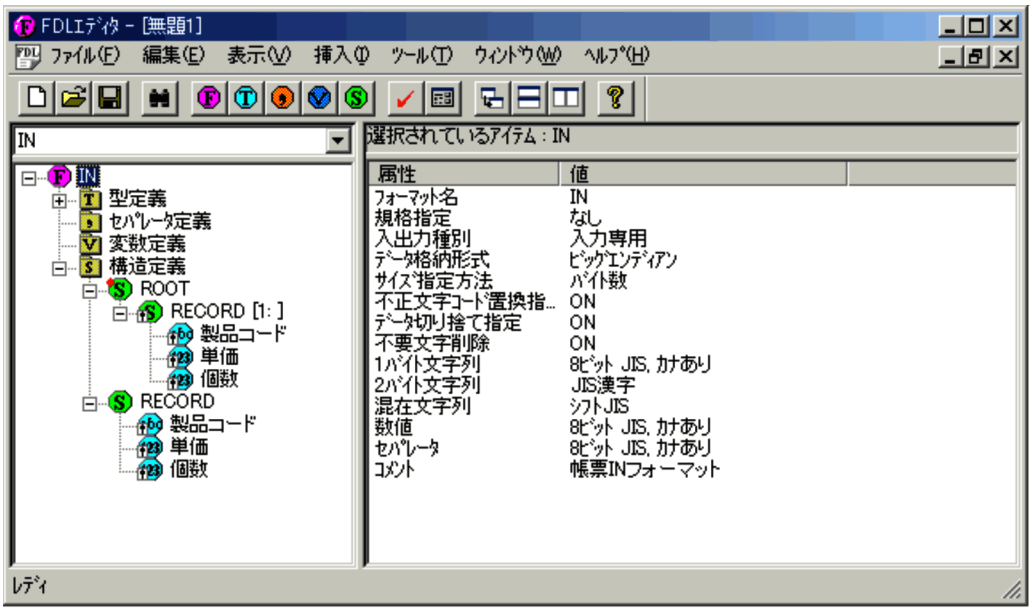
「単価」「個数」も同様に構造「RECORD」にドラッグ＆ドロップします。

最後に，構造「RECORD」は構造「ROOT」の子コンポーネントとします。構造「RECORD」を構造「ROOT」にドラッグ＆ドロップし，［操作選択］ダイアログで構造「ROOT」の子コンポーネントとして定義します。



また，レコードの個数は不定なので，［コンポーネントのプロパティ］ダイアログで，コンポーネント最大出現数を省略して定義します。

以上で帳票「IN」のフォーマットは完成です。



(f) フォーマットの検証

チェックコマンドを実行して，フォーマットの定義内容にエラーがないことを確認します。

(g) FDL ファイルの保存

FDL に名前を付けて保存します。ここでは，ファイル名は「IN」とします。

次に，帳票「OUT」を定義します。

(2) 帳票「OUT」のフォーマット

帳票「OUT」を次に示します。

図 2-3　帳票「OUT」

帳票「OUT」のイメージ

	製品コード	希望価格(円)	数量	小計金額(円)
1	ABC-123-E	56,800	50	2,840,000
2	DEF-456-J	19,800	163	3,227,400
:	:	:	:	:
N	XYZ-789-S	3,900	2	7,800

帳票「OUT」のデータフォーマット

製品コード，希望価格，数量，小計金額（改行）
レコード
⋮
レコード

帳票「OUT」のレコードは，「製品コード」「希望価格」「数量」「小計金額」の四つの要素で構成されています。四つの要素はすべて可変長です。レコード内の四つの要素間は「,」で区切り，レコードの終わりには改行コードを入れます。レコードの個数は不定です。

図 2-3 の帳票「OUT」のフォーマットを定義します。

(a) フォーマット名の定義

帳票「IN」のフォーマット定義と同様に，フォーマットを新規作成して定義します。

(b) 型の定義

帳票「OUT」には，「製品コード」「希望価格」「数量」「小計金額」という四つの要素があります。帳票「IN」の型の定義と同様に，型を定義します。［型のプロパティ］ダイア

ログで型名や属性を，［型の詳細］ダイアログで使用する文字コードやサイズを定義します。

(c) セパレータの定義

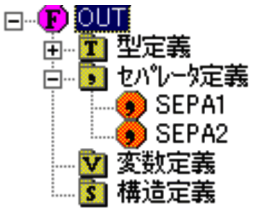
帳票「OUT」には，レコード内の要素を区切るセパレータと，レコードの終わりを表すセパレータの 2 種類のセパレータを使用しています。

セパレータ挿入コマンドを実行して，フォーマットにセパレータを挿入します。

［セパレータのプロパティ］ダイアログに，セパレータの名前や値を入力します。

セパレータの定義が済んだら，ダイアログを閉じます。

セパレータが作成されます。



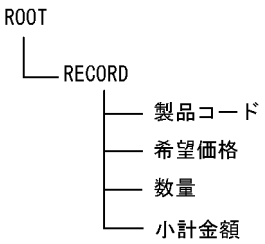
(d) 変数の定義

帳票「OUT」では変数を使用しないので，ここでは定義しません。

次に，構造を定義します。

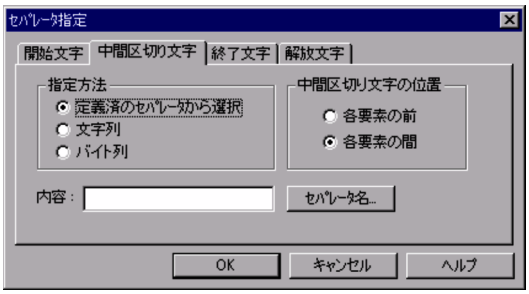
(e) 構造の定義

帳票「OUT」のデータの構造をツリーで表すと次のようになります。



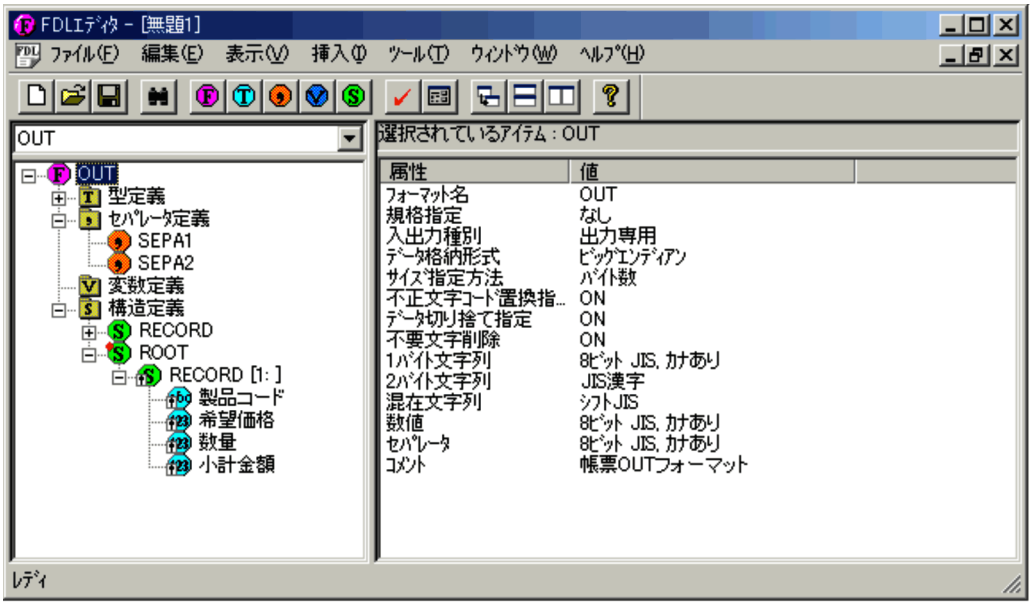
構造は帳票「IN」と同様に作成します。次に，コンポーネントで使用するセパレータの設定について説明します。

帳票「OUT」で使用する 2 種類のセパレータは，どちらも構造「RECORD」に対して設定します。構造「RECORD」のプロパティで，［セパレータ指定］ダイアログを表示します。



ここでは，セパレータの名前や値を定義したので，セパレータ名を指定します。中間区切り文字として「,」のセパレータ，終了文字として改行コードのセパレータを指定します。

以上で帳票「OUT」のフォーマットは完成です。



チェックコマンドを実行して，フォーマットの定義内容にエラーがないことを確認し，FDL に名前を付けて保存します。ここでは，ファイル名は「OUT」とします。

以上で帳票「IN」と帳票「OUT」のデータのフォーマットを定義した FDL ファイルが完成しました。

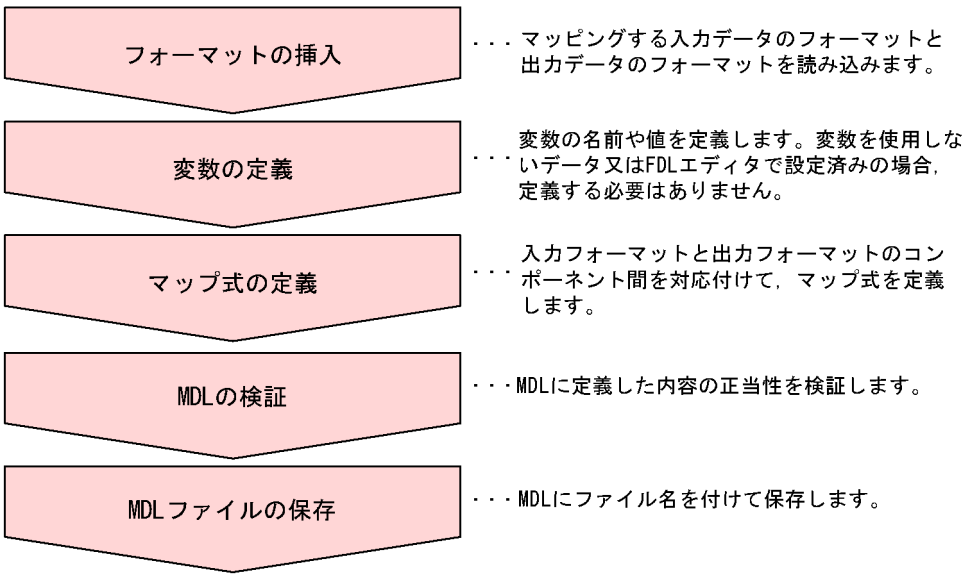
2.2　マッピング

この節では，FDL エディタで定義した入力データのフォーマットと出力データのフォーマットのコンポーネント間を対応付けるマッピングについて説明します。

コンポーネントをドラッグ＆ドロップしマッピングして，コンポーネント間を対応付ける式を MDL エディタ上で定義します。なお，この式を「マップ式」と呼びます。

MDL エディタでのマッピングの手順を次に示します。

図 2-4　マッピングの手順



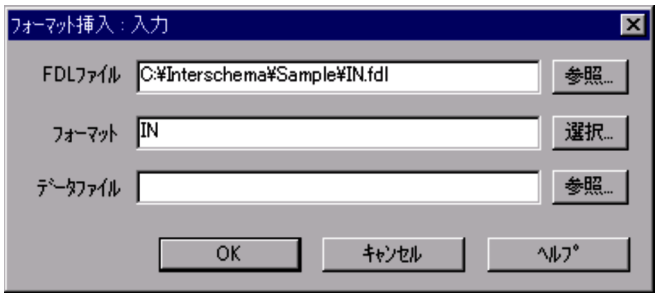
例を使用してマッピングについて説明します。なお，以下のマッピング定義例は，マッピングの流れを理解しやすくすることを意図したものなので，マッピング定義の詳細説明は省略しています。マッピング定義の詳細については，「4.　MDL エディタの操作」を参照してください。

「2.1.3　フォーマットの定義手順」で定義した帳票「IN」のフォーマットと帳票「OUT」のフォーマットをマッピングします。

(a) フォーマットの挿入

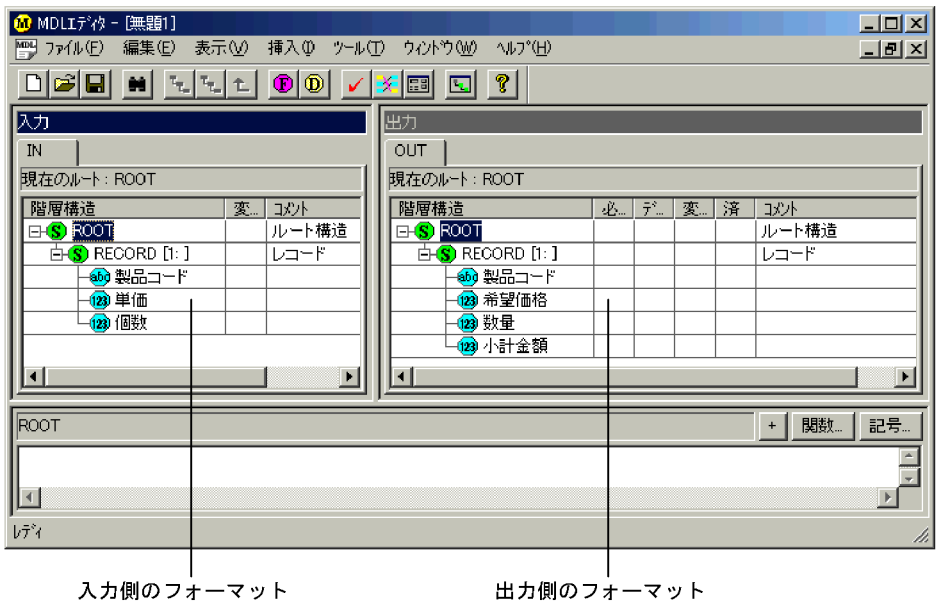
マッピングする入力側と出力側のフォーマットを読み込みます。ここでは，入力側のフォーマットとして帳票「IN」のフォーマット「IN」を，出力側のフォーマットとして帳票「OUT」のフォーマット「OUT」を読み込みます。

フォーマット挿入コマンドを実行すると［フォーマット挿入］ダイアログが表示されます。



入力データ又は出力データのファイル名が決まっている場合は，このダイアログにデータファイル名を指定します。ファイル名が決まっていない場合は，データ変換時のコマンド引数に，ファイル名を指定することもできます。例では，データ変換時のコマンド引数にファイル名を指定するので，データファイル名を指定しません。

挿入したフォーマットは，ツリー構造でルート構造以下が表示されます。



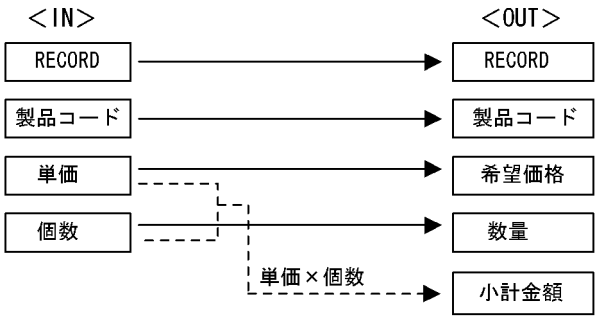
(b) 変数の定義

この定義例では変数を使用しないため，変数の定義はしません。

(c) マップ式の定義

入力データのコンポーネントと出力データのコンポーネント間をマッピングします。

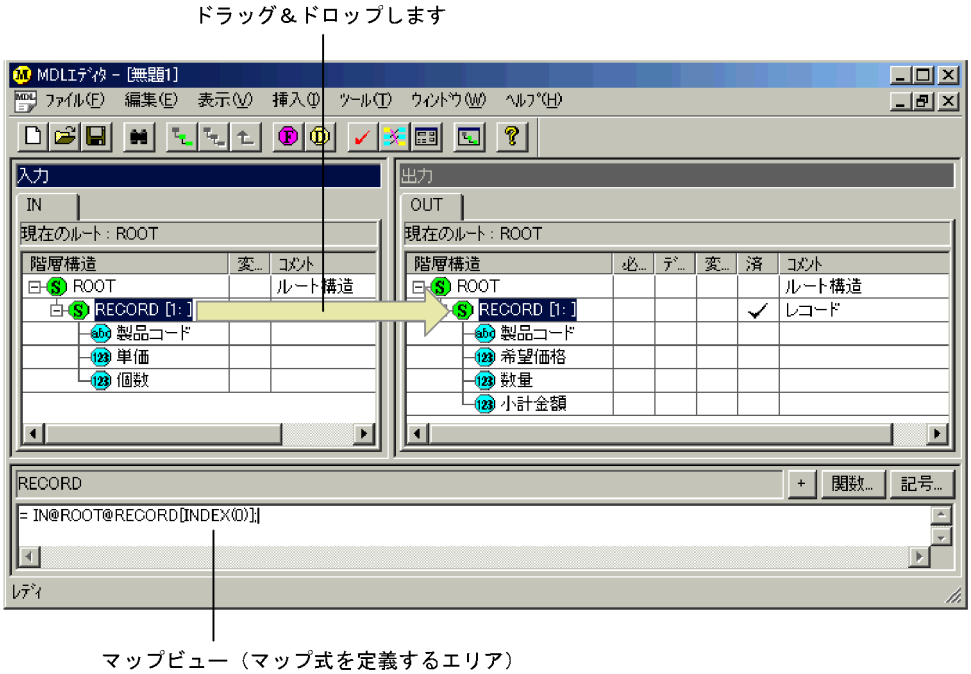
帳票「IN」と帳票「OUT」のコンポーネントの対応関係は次のようになります。



帳票「OUT」の要素「RECORD」「製品コード」「希望価格」「数量」は，対応する帳票「IN」の要素の値を代入します。帳票「OUT」の要素「小計金額」は，帳票「IN」に対応する要素がないので，値を計算して出力するようにマップ式を定義します。

マッピングは，出力データのコンポーネントに対して定義します。出力データのコンポーネントに，対応する入力データのコンポーネントをドラッグ＆ドロップし，マップ式を定義します。

この例では，出力側のコンポーネント「RECORD」「製品コード」「希望価格」「数量」に，対応する入力側のコンポーネントをドラッグ＆ドロップして代入します。

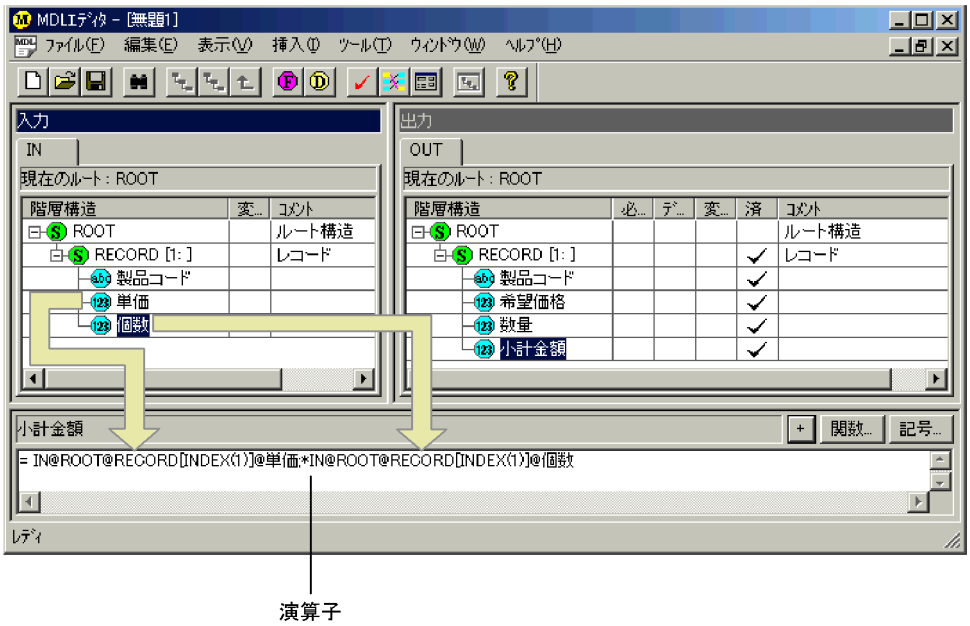


マップビューには，選択している出力側のコンポーネントに定義されているマップ式が表示されます。コンポーネントをドラッグ＆ドロップすると，マップビューにマップ式

の代入式が挿入されます。

マップ式には，コンポーネント名，演算子，関数及び記号を使用できます。出力側のコンポーネントを選択し，マップビューへコンポーネントをドラッグ＆ドロップすると，ドラッグ＆ドロップしたコンポーネント名がマップビューに記述されます。演算子，関数及び記号はダイアログで選択して使用できます。コンポーネント名，関数，演算子及び記号は，マップビューに直接テキストで入力することもできます。マップ式の詳細については，「6.6.6　マップ式」を参照してください。

出力側の要素「小計金額」は，入力側の二つの要素「単価」「個数」の乗算で求めるように演算子を使用してマップ式を定義します。コンポーネント「小計金額」を選択し，マップビューに二つの要素をドラッグ＆ドロップし，乗算の演算子を使用します。



これでマッピングの定義は完成です。

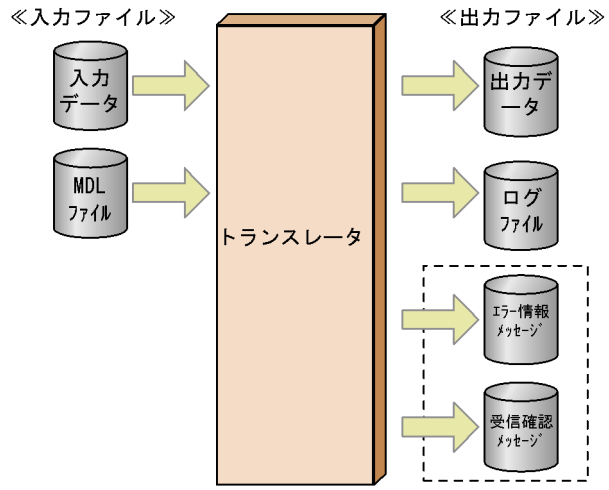
チェックコマンドを実行して，マッピングの内容にエラーがないことを確認し，MDLファイルに名前を付けて保存します。ここでのファイル名は「SAMPLE.mdl」とします。

2.3　データの変換

この節では，データの変換について説明します。

トランスレータがデータを変換するための情報を定義した MDL ファイルを使用して，データを変換します。トランスレータでデータを変換するために必要なファイルは，変換する入力データのファイルと MDL ファイルです。トランスレータでのデータの変換を次に示します。

図 2-5　データの変換



（凡例）　点線内は，入力データがCII標準形式の場合に指定できるオプションです。

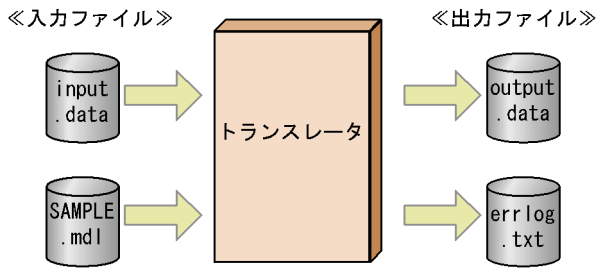
トランスレータは，コマンド「ettrans」を実行して起動します。コマンドの引数として，MDL ファイルを指定すると，MDL ファイルのマッピング情報を基に入力データを変換し，出力データのファイルに出力します。MDL ファイルに入力データ及び出力データのファイル名を定義していない場合，又は定義していないファイル名で変換したい場合は，コマンドの引数としてファイル名を指定できます。

また，トランスレータのデータの変換結果として，ログファイルに戻り値やエラーメッセージが出力されます。コマンドの引数として，ログファイル名を指定すれば，ユーザが指定したファイルにログを出力できます。

次に，これまでフォーマット及びマッピングを定義した帳票「IN」，並びに帳票「OUT」のデータを変換します。データの変換に必要なファイル名を次に示します。

- MDL ファイル名：SAMPLE.mdl
- 入力データのファイル名：input.data
- 出力データのファイル名：output.data

ログファイル名は指定しないので，トランスレータが設定する標準のファイル名（errlog.txt）となります。



ここでは，コマンド実行時に引数として入力データと出力データのファイル名を指定します。コマンドは次のようになります。

```
ettrans  SAMPLE.mdl -F IN input.data OUT output.data
```

「SAMPLE.mdl」に定義された変換情報に従って，「input.data」ファイルのデータが変換され，「output.data」ファイルに出力されます。また，「errlog.txt」にログが出力されます。コマンドの詳細については，「5. 変換のコマンド」を参照してください。

入力データがファイルではなくメモリ上にある場合，Interschema が提供するデータ変換処理の API を使用したプログラムで，トランスレータを実行してデータを変換してください。データ変換処理 API の概要については，「8. インタフェース」を参照してください。データ変換処理 API の詳細については，「9. データ変換処理 API（C 言語）」，又は「10. データ変換処理 API（Java 言語）」を参照してください。

3 FDL エディタの操作

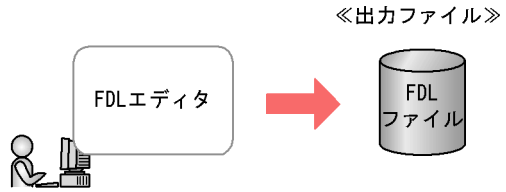
この章では，FDL エディタの操作の概要，具体例を使用した FDL エディタでの操作方法について説明します。

3.1 操作の概要

この節では，FDL ファイルを作成するための FDL エディタの操作手順，FDL エディタの画面構成，FDL エディタのコマンドについて説明します。

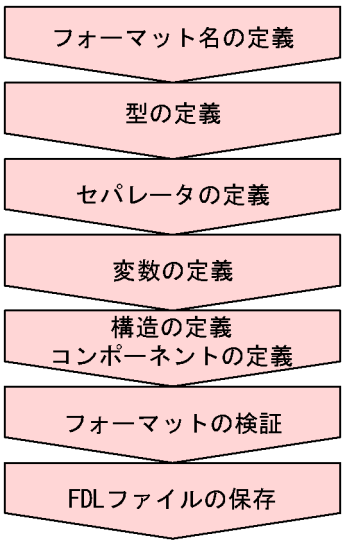
3.1.1 FDL ファイルの作成手順

FDL エディタでは，入力データ，出力データのデータの構造に基づいてフォーマットを作成します。定義したフォーマットをファイルとして保存したものを FDL ファイルと呼びます。



FDL ファイルの作成手順を次に示します。

図 3-1 FDL ファイルの作成



手順について次に説明します。

1. **フォーマット名の定義**
   作成するフォーマットの名前を付けたり，フォーマット全体に対するプロパティを定義したりします。一つの FDL ファイルには，複数のフォーマットを定義できます。

2. **型の定義**
   変換するデータの要素の属性やサイズを定義します。
3. **セパレータの定義**
   構造内で使用するセパレータに名前を付けて値を定義します。構造内でセパレータを使用しないデータの場合，セパレータを作成する必要はありません。また，構造を定義する場合に直接セパレータの値を指定するときも，作成する必要はありません。
4. **変数の定義**
   変換時に使用する変数の名前や値を定義します。変数を使用しないデータの場合，変数を定義する必要はありません。
5. **構造の定義**
   変換するデータの構造やコンポーネントを定義します。
6. **フォーマットの検証**
   チェックコマンドで，フォーマットの定義内容の正当性を検証します。MDL エディタにフォーマットを読み込んでマッピングする前に，チェックコマンドを実行してフォーマットの定義内容にエラーがないか確認する必要があります。
7. FDL **ファイルの保存**
   FDL に名前を付けて保存します。これで FDL ファイルが作成できます。FDL ファイルの拡張子は，「.fdl」です。

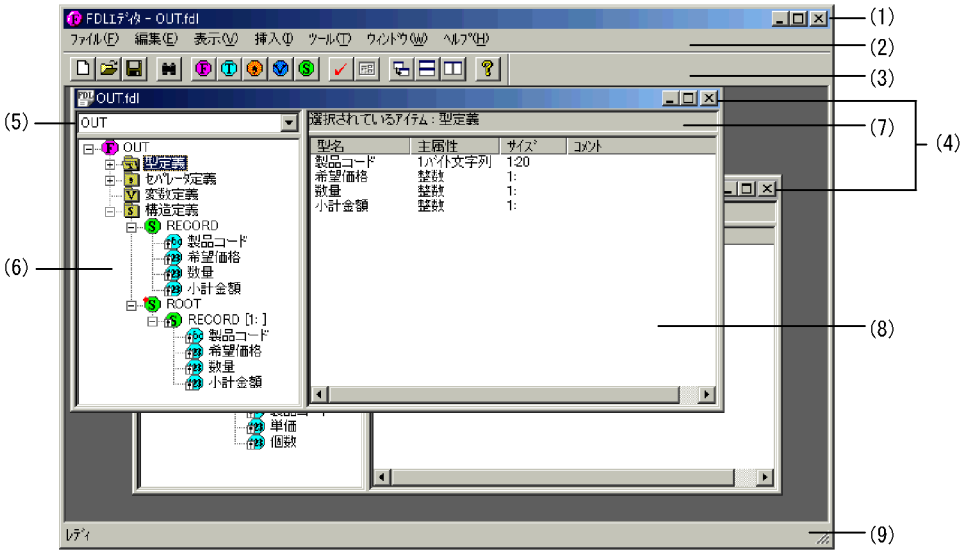
3.1.2 FDL エディタの画面構成

ここでは，FDL エディタの画面構成について説明します。FDL エディタは次の操作で起動できます。

1. **［スタート］－［プログラム］－［**Interschema**］－［**FDL **エディタ］を選択します**
   FDL エディタが起動します。FDL エディタは，同時に複数の FDL ファイルを開くことができます。

FDL エディタの画面構成を次に示します。図中の番号は，説明の番号と一致しています。

図 3-2　FDL エディタの画面構成



(1) FDL エディタウィンドウ

FDL エディタのメインとなるフレームウィンドウです。

(2) メニューバー

実行するコマンドを指定するためのメニューです。コマンドの詳細については，FDL エディタのヘルプを参照してください。

(3) ツールバー

使用頻度の高いコマンドのボタンを集めたバーです。マウスでドラッグすると，任意の位置に移動できます。各ボタンの上にマウスカーソルを移動させると，ステータスバーにボタンの説明が表示されます。

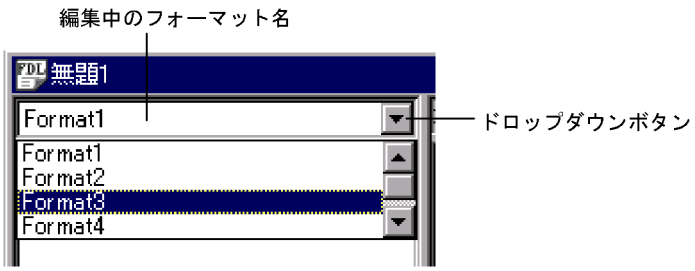
(4) ドキュメントウィンドウ

一つの FDL ドキュメントに対応するウィンドウです。複数のドキュメントを編集できます。また，一つのドキュメントに対して複数のドキュメントウィンドウを開くことができます。ただし，FDL エディタで一度に編集できるドキュメントは 20 個までです。

(5) フォーマット指定コンボボックス

FDL 内に定義されているフォーマットのうち，編集中のフォーマットを表示するためのドロップダウンリスト形式のコンボボックスです。

図 3-3 フォーマット指定コンボボックス

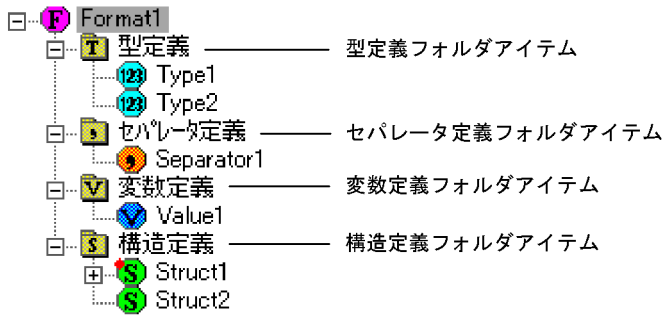


ドロップダウンボタンをクリックして，編集したいフォーマットを切り替えることができます。

(6) ツリービュー

フォーマット内の型，セパレータ，変数，構造，コンポーネントのデータを階層的に表示するためのビューです。ツリーを構成する要素を「アイテム」と呼びます。データを定義すると定義フォルダアイテムの下に定義したデータのアイテムが作成されます。また，アイテムをドラッグ＆ドロップして，データのコピーや移動ができます。

図 3-4 ツリービュー



ツリーは次の動作によって縮小・展開できます。

- アイテムをダブルクリックする
- アイテムの左側に表示されている「＋」マーク又は「－」マークをクリックする

(7) セレクト情報

ツリービュー内で選択されているアイテム名を表示するエリアです。

(8) リストビュー

ツリービュー内で選択されているアイテムの情報をリスト形式で表示するビューです。リストビュー内に表示されているアイテムを，ツリービューにドラッグ＆ドロップして，コンポーネントの作成，データのコピーや移動ができます。

(9) ステータスバー

メニューバーやツールバーからコマンドを選択するときに，コマンドの説明を表示するエリアです。

3.1.3 FDL エディタのコマンド一覧

FDL エディタのコマンドの一覧を表 3-1 に示します。

表 3-1 FDL エディタのコマンド一覧

メニュー名	メニューアイテム名	ショートカットキー	説明
ファイル (F)	新規作成 (N)	［Ctrl］＋［N］	FDL ドキュメントを新規に作成します。
	開く (O)...	［Ctrl］＋［O］	FDL ファイルを読み込みます。
	閉じる (C)	－	FDL ドキュメントウィンドウを閉じます。
	上書き保存 (S)	［Ctrl］＋［S］	FDL ドキュメントの内容を対応する FDL ファイルに上書き保存します。
	名前を付けて保存 (A)...	－	FDL ドキュメントに名前を付けて，FDL ファイルとして保存します。
	外部出力 (E)...	－	フォーマット情報を出力します。
	プロパティ (P)...	－	FDL ドキュメントのプロパティを表示します。
	（ファイル名）	－	最近使用したファイル（読み込んだファイル又は保存したファイル）を読み込みます。
	FDL エディタの終了 (X)	－	FDL エディタを終了します。
編集 (E)	削除 (D)	［Delete］	ツリービュー，リストビュー内で選択したデータを削除します。
	検索 (F)...	［Ctrl］＋［F］	ツリービュー内で指定したデータ又は式文字列を検索します。
表示 (V)	ツールバー (T)	－	ツールバーの表示／非表示を切り替えます。
	ステータスバー (S)	－	ステータスバーの表示／非表示を切り替えます。
挿入 (I)	フォーマット (F)...	－	FDL ドキュメントに新規フォーマットを追加します。
	型 (T)...	－	選択しているフォーマットに新規の型を追加します。
	セパレータ (E)...	－	選択しているフォーマットに新規のセパレータを追加します。
	変数 (V)...	－	選択しているフォーマットに新規の変数を追加します。

メニュー名	メニュー アイテム名	ショートカット キー	説明
	構造 (S)...	―	選択しているフォーマットに新規の構造を追加します。
ツール (T)	チェック (C)...	―	フォーマットの定義内容の正当性を検証します。
	プロパティ (P)...	―	指定しているデータの詳細情報を表示します。
	オプション (O)...	―	FDL エディタのオプションを設定します。
ウィンドウ (W)	新しいウィンドウを開く (N)	―	作業中のドキュメントのウィンドウをもう一つ開きます。
	重ねて表示 (C)	―	ドキュメントウィンドウのタイトルが重ならないように整列します。
	上下に並べて表示 (H)	―	ドキュメントウィンドウを上下に並べます。
	左右に並べて表示 (V)	―	ドキュメントウィンドウを左右に並べます。
	(ドキュメント名)	―	指定した FDL ドキュメントのウィンドウをアクティブにします。
ヘルプ (H)	目次 (C)...	―	FDL エディタのヘルプの目次を表示します。
	検索 (S)...	―	FDL エディタのヘルプのトピック一覧を表示します。
	バージョン情報 (A)...	―	FDL エディタの形名，バージョンなどを表示します。

(凡例)

―：ショートカットキーはありません。

3.2 簡単なデータ構造のフォーマット定義例

この節では，簡単なデータ構造のフォーマットを定義しながら，FDL エディタの操作について説明します。

固定長形式のフォーマットの中でも，簡単なデータ構造のメッセージのフォーマットを定義します。メッセージのデータ構造を図 3-5 に示します。

図 3-5 メッセージのデータ構造

メッセージのイメージ

注文番号 00001　　注文日 981030
単位 1 (個)
品名 ABC部品

No	単価(円)	個数	金額(円)
1	1000	100	100000
2	2000	200	200000
3	10000	50	500000

データフォーマットのイメージ

注文番号	単位	単価	個数	金額	注文日	品名
整数 5	文字列 1	整数 5＊3個	整数 5＊3個	整数 10＊3個	日付 6	文字列 80

（凡例） 数値はバイト数です。

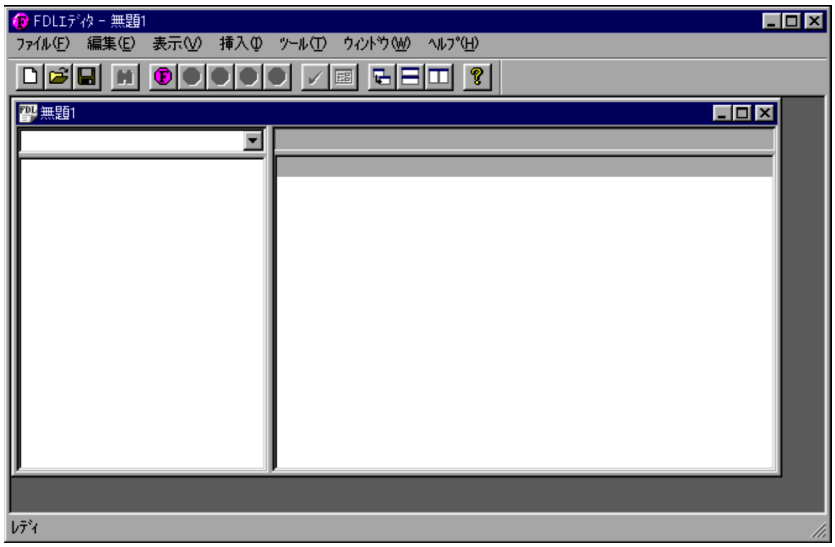
このメッセージは，固定長の項目が連続して出現し，「単価」「個数」「金額」はそれぞれ 3 個ずつ連続して出現する注文メッセージです。作成するフォーマット名は，「FIX1」とします。

3.2.1 フォーマット名の定義

まず，定義したフォーマットを格納するためのドキュメントを作成します。

1. **［ファイル］－［新規作成］を選択します**
   ドキュメントウィンドウが表示されます。

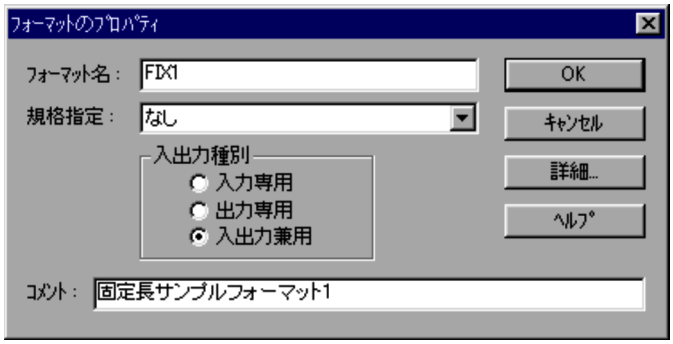
図 3-6 ドキュメントウィンドウ



次に，ドキュメントウィンドウ内にフォーマットを新規作成して，フォーマット名などのプロパティを定義します。

2. **［挿入］－［フォーマット］を選択します**
［フォーマットのプロパティ］ダイアログが表示されます。作成するフォーマット全体に対するプロパティを定義します。

図 3-7 ［フォーマットのプロパティ］ダイアログ



ダイアログの設定内容は次のとおりです。

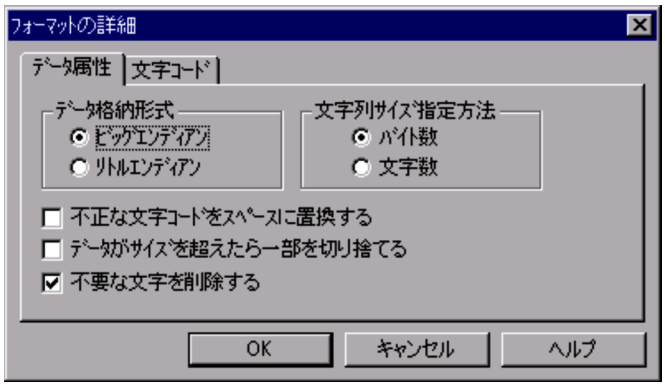
- フォーマット名
作成するフォーマットの名前を入力します。ここでは「FIX1」とします。
- 入出力種別
フォーマットの入出力種別は，データの特性によって使い分けます。固定長データの場合は，入力データと出力データの構造にほとんど差がないので，ここでは「入出力兼用」を選択します。
- コメント
フォーマットの対応するデータについて，簡単にコメントを記述します。コメント

はデータに直接関係するものではありません。型及び構造のプロパティでのコメントについても同様です。

次に，フォーマットの全体の詳細内容を設定します。

3. **［フォーマットのプロパティ］ダイアログで［詳細］ボタンをクリックします**
［フォーマットの詳細］ダイアログが表示されます。このダイアログには，［データ属性］タブと［文字コード］タブの 2 種類のタブがあります。各タブの設定内容について次に説明します。

図 3-8 ［フォーマットの詳細］ダイアログの［データ属性］タブ



ダイアログの設定内容は次のとおりです。

- データ格納形式
このメッセージで扱うデータのエンディアンの指定は，2 進数や Unicode の文字列のバイト順を示すものです。2 進数を扱わないデータでは特に意識する必要はないので，ここでは「ビッグエンディアン」を指定します。
- 文字列サイズ指定方法
このメッセージで扱うデータはすべて固定サイズなので，サイズ計算方法は「バイト数」を指定します。
- 不正な文字コードをスペースに置換する
不正な文字コードがあった場合は常にエラーとするように，ここでは選択しません。選択した場合は，変換実行時のオプションで，エラーとするかどうか動的に変更できます。
- データがサイズを超えたら一部を切り捨てる
出力時のデータが規定したサイズを超えた場合は常にエラーとなるように，ここでは選択しません。選択した場合は，変換実行時のオプションで，エラーとするかどうか動的に変更できます。
- 不要な文字を削除する
入力データに対しては埋め字をデータと見なされないようにするため，ここでは選択します。

図 3-9 ［フォーマットの詳細］ダイアログの［文字コード］タブ

ダイアログの設定内容は次のとおりです。

- 1 バイト文字列

  このメッセージでは，要素「品名」以外に使用する文字は 1 バイトコードなので，初期値の 1 バイト文字列には「8 ビット JIS，カナあり」を指定します。各要素の文字コードの初期設定値がこのダイアログで指定した文字コードと同じであれば，後述する要素の型定義で文字コードを宣言する必要はありません。

上記以外は，初期設定のままとします。

4. **文字コードを指定したら，［フォーマットの詳細］ダイアログで［OK］ボタンをクリックします**

   ［フォーマットの詳細］ダイアログが閉じて，［フォーマットのプロパティ］ダイアログに戻ります。

5. **［フォーマットのプロパティ］ダイアログで［OK］ボタンをクリックします**

   ［フォーマットのプロパティ］ダイアログが閉じて，ツリービューに「FIX1」というアイテム名のフォーマットが作成されます。フォーマットのアイテムの下には，次に示す四つのフォルダアイテムが作成されます。

次に，型を定義します。

3.2.2 型の定義

メッセージ内で使用する型を定義します。メッセージデータの要素を見ると，数値要素と文字列要素があります。

(1) 数値要素「注文番号」「単価」「個数」の定義

まず，数値要素のうち，要素「注文番号」を定義します。

フォーマットに型を新規作成します。

1. **［挿入］－［型］を選択します**
   ［型のプロパティ］ダイアログが表示されます。

図 3-10 ［型のプロパティ］ダイアログ

ダイアログの設定内容は次のとおりです。

- 型名
  作成する型の名前を入力します。ここでは「注文番号」とします。
- データ種別
  型のデータ種別を指定します。ここでは「数値」とします。
- 主属性
  型の主属性を指定します。ここでは「整数」とします。

次に，型の詳細を設定します。

2. **［型のプロパティ］ダイアログで［詳細］ボタンをクリックします**
   ［型の詳細］ダイアログが表示されます。このダイアログは，［型のプロパティ］ダイアログの主属性に何を指定したかによって，表示されるタブの種類が異なります。ここでは，主属性に「整数」を指定したので，次のタブが表示されます。

図 3-11 ［型の詳細］ダイアログの［数値属性］タブ

ダイアログの設定内容は次のとおりです。

- 文字コード
  ここでは，［フォーマットの詳細］ダイアログで指定した「8 ビット JIS」を使用するので，「フォーマットの指示に従う」を指定します。
- 左右寄せ
  データが要素のサイズに満たない場合に，要素のエリア内で左右どちらに詰めて出力するかを指定します。一般的に，数値要素はエリアに対して右詰めで出力します。ここでは「右寄せ」を指定します。
- 埋め字
  データが要素のサイズに満たない場合に，余った部分を埋める文字を指定します。一般的に，数値要素はエリアに対して右詰めで，埋め字に「0」を使用します。ここでは「0」を指定します。
- サイズ
  「固定」，「5」バイト
- 桁数－全体
  「5」桁

上記以外は，初期値のままとします。

3. **型の詳細を設定したら，［OK］ボタンをクリックします**
   ［型の詳細］ダイアログを閉じて［型のプロパティ］ダイアログに戻ります。
4. **［型のプロパティ］ダイアログで［OK］ボタンをクリックします**
   ［型のプロパティ］ダイアログが閉じて，型が作成されます。ツリービューの型定義フォルダの下に「注文番号」というアイテム名の型が作成されます。

ほかの数値要素「単価」「個数」も型を定義します。型名とコメント以外はすべて，要素「注文番号」と同じです。

(2) 文字列要素「単位」の定義

次に，文字列要素のうち，要素「単位」を定義します。

要素「単位」には，値とその意味が対応付けられているので，決まった文字だけをデータとして指定します。この場合，要素で扱う値をあらかじめ定義し，定義している値以外は扱わないように指定できます。要素「単位」の値は，1 バイト文字列型とします。

［型のプロパティ］ダイアログで次のように設定します。

- 型名
  「単位」
- データ種別
  「文字列」
- 主属性
  「1 バイト文字列」

［型の詳細］ダイアログで次のように設定します。

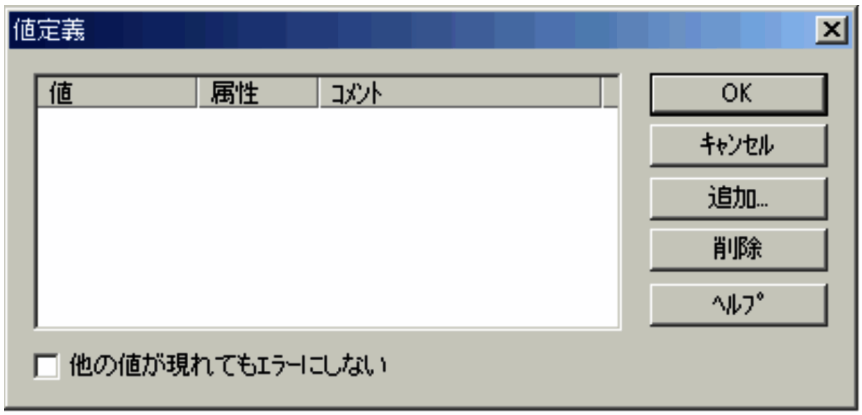
- 文字コード
  「フォーマットの指示に従う」
- 左右寄せ
  「左寄せ」
- 埋め字
  「文字」を指定します。ここでは，埋め字にスペースを入力します。
- サイズ
  「固定」,「1」バイト

上記以外は，初期値のままとします。

次に，型で扱う値（単位）を定義します。

1. **要素「単位」の［型のプロパティ］ダイアログで［値定義］ボタンをクリックします**
   ［値定義］ダイアログが表示されます。このダイアログの値リストには，その要素で扱える値が一覧表示されます。ここでは，値を定義していないので値リストには何も表示されません。

図 3-12 ［値定義］ダイアログ



2. **［値定義］ダイアログで［追加］ボタンをクリックします**
［値追加］ダイアログが表示されます。

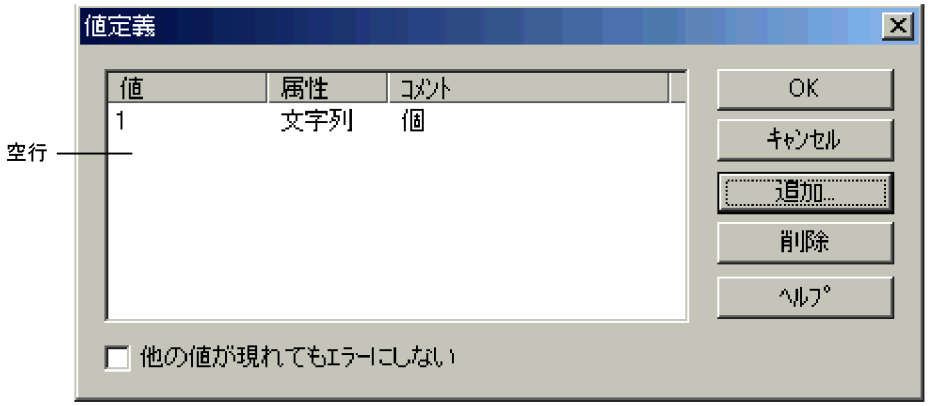
図 3-13 ［値追加］ダイアログ

要素で扱える値を定義します。この例では，値として「1」(個)，「2」(メートル)，「3」(キログラム)を定義します。この要素は1バイト文字列型とします。まず，値として「1」(個)を定義します。

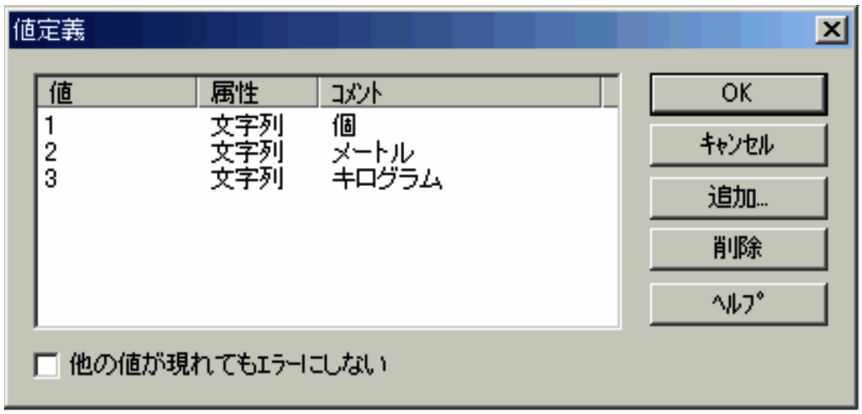
- 値
「1」
- 値の属性
「文字列」
- コメント
値の意味について簡単なコメントを記述します。ここでは「個」とします。

3. **［OK］ボタンをクリックします**
［値追加］ダイアログを閉じて，［値定義］ダイアログに戻ります。
定義した値が［値定義］ダイアログの値リストに表示されます。

図 3-14 ［値定義］ダイアログ（定義例）



同様に，ほかの値も追加します。追加した値が値リストのどこに追加されるかは，追加する前に選択していた値リストの場所によって異なります。定義した値を選択していたら，その値の一つ前に追加され，空行を選択していたらリストの最後に追加されます。次に，空行を選択して値を追加した場合の［値定義］ダイアログを示します。



［値定義］ダイアログの「他の値が現れてもエラーにしない」は選択しないままにします。こうしておくと，要素「単位」は，［値定義］ダイアログに定義されている値以外の値を扱いません。

4. **値を定義したら，［値定義］ダイアログの［OK］ボタンをクリックします**
［値定義］ダイアログを閉じて［型のプロパティ］ダイアログに戻ります。
［型のプロパティ］ダイアログで［OK］ボタンをクリックすると，ダイアログが閉じてツリービューに型が作成されます。

(3) 要素「金額」の定義

次に，要素「金額」を定義します。

［型のプロパティ］ダイアログで次のように設定します。

- 型名

「金額」
- データ種別
「数値」
- 主属性
「整数」

［型の詳細］ダイアログで次のように設定します。
- 文字コード
「フォーマットの指示に従う」
- 左右寄せ
「右寄せ」
- 埋め字
「0」
- サイズ
「固定」,「10」バイト
- 桁数－全体
「10」桁

上記以外は，初期値のままとします。

(4) 要素「注文日」の定義

次に，要素「注文日」を定義します。

［型のプロパティ］ダイアログで次のように設定します。
- 型名
「注文日」
- データ種別
「日付時刻」
- 主属性
「日付」

［型の詳細］ダイアログで次のように設定します。この場合，表示されるタブは 2 種類あります。

［テキスト属性］タブ
- 文字コード
「フォーマットの指示に従う」
- 左右寄せ
「右寄せ」
- 埋め字
「文字」を指定します。ここでは，埋め字に「0」を指定します。
- サイズ
「固定」,「6」バイト

［日付属性］タブ

- 形式
  日付のデータ形式を指定します。ここでは，最大 6 桁の数字で日付を表す「YYMMDD」を指定します。

(5) 要素「品名」の定義

次に，要素「品名」を定義します。

［型のプロパティ］ダイアログで次のように設定します。

- 型名
  「品名」
- データ種別
  「文字列」
- 主属性
  要素「品名」は文字列中に漢字を含むので，文字列中に 1 バイト文字と 2 バイト文字を混在して使用できる「混在文字列」を指定します。

［型の詳細］ダイアログで次のように設定します。

- 文字コード
  フォーマットに対して指定した文字コードと異なる文字コードを指定した場合，要素に対する指定が有効となります。ここでは，要素「品名」は漢字を含む文字列なので「シフト JIS」を指定します。
- 左右寄せ
  「左寄せ」
- 埋め字
  「文字」を指定します。ここでは，埋め字にスペースを入力します。
- サイズ
  「固定」,「80」バイト

上記以外は，初期値のままとします。

以上で各要素が定義できました。

各要素の定義内容を表 3-2 に示します。

表 3-2 型の定義

型名	データ種別／主属性	文字コード	左右寄せ／埋め字	サイズ	備考
注文番号	数値／整数	フォーマットの指示に従う	右 0	固定 5	桁数：5
単価					
個数					
単位	文字列／1バイト文字列型	（8ビットJIS）	左 スペース	固定 1	値の定義 • 1 コメント：個 • 2 コメント：メートル • 3 コメント：キログラム エラー検証なし 値の意味は文字列
金額	数値／整数		右 0	固定 10	－
注文日	日付時刻／日付		右 0	固定 6	データ形式：「YYMMDD」
品名	文字列／混在文字列	シフトJIS	左 スペース	固定 80	－

（凡例）

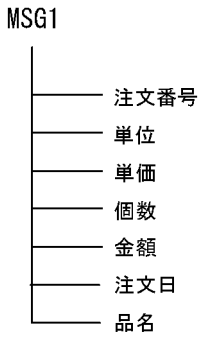
サイズ：サイズの単位はバイトです。

フォーマットの指示に従う：［フォーマットの詳細］ダイアログで設定した文字コードに従うという意味です。

3.2.3 構造の定義

次に，定義した要素の型を使用して，フォーマット「FIX1」の構造を定義します。フォーマット「FIX1」の構造を図 3-15 に示します。

図 3-15 フォーマット「FIX1」の構造



メッセージ「MSG1」は7種類の要素から構成されています。要素は定義した順番に出

現する逐次構造です。

まず，変換するデータの最上位のルート構造「MSG1」を定義します。

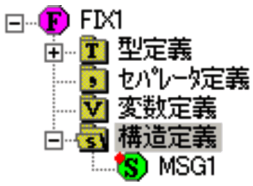
1. **［挿入］－［構造］を選択します**
   ［構造のプロパティ］ダイアログが表示されます。

図 3-16 ［構造のプロパティ］ダイアログ

ダイアログの設定内容は次のとおりです。

- 構造名
  作成する構造の名前を入力します。ここでは「MSG1」とします。
- 構造の種類
  「MSG1」は要素が定義順に現れる構造なので，ここでは「逐次構造」とします。
- ルートに指定する
  作成する構造をルート構造に指定する場合にチェックを入れます。ここでは，
  チェックを入れます。

2. **［OK］ボタンをクリックします**
   ［構造のプロパティ］ダイアログが閉じて，ツリービューの構造定義フォルダアイテムの下にルート構造「MSG1」が作成されます。



ルート構造を作成したら，ルート構造の下に要素をドラッグ＆ドロップします。ドラッグ元はツリービューでも，リストビューでもかまいません。逐次構造では，定義した順でコンポーネントが出現します。また，コンポーネントの並び順がデータの並び順になるので，順序を間違えないように注意してください。ここでは，次の構造のとおりにコンポーネントが出現するようにします。

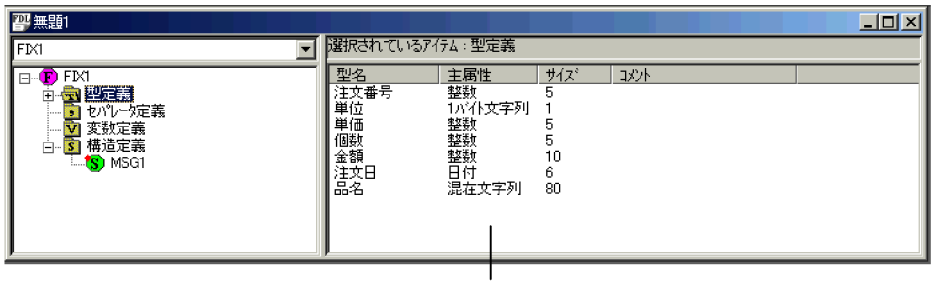
MSG1

- 注文番号［1］
- 単位［1］
- 単価［3］
- 個数［3］
- 金額［3］
- 注文日［1］
- 品名［1］

（凡例）［ ］内は出現回数です。

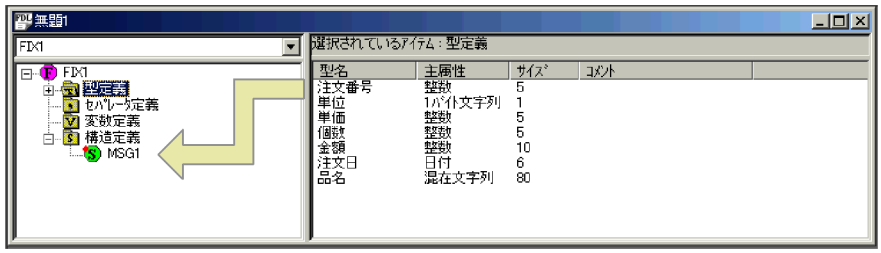
各定義フォルダアイテムを選択すると，リストビューにフォルダ内のデータが表示されます。リストビューからツリービューへデータをドラッグ＆ドロップすると容易に定義できます。
ここでは，型定義フォルダアイテムから要素をドラッグ＆ドロップします。

3. **ツリービューの型定義フォルダアイテムを選択します**
   リストビューに型を定義した要素が表示されます。



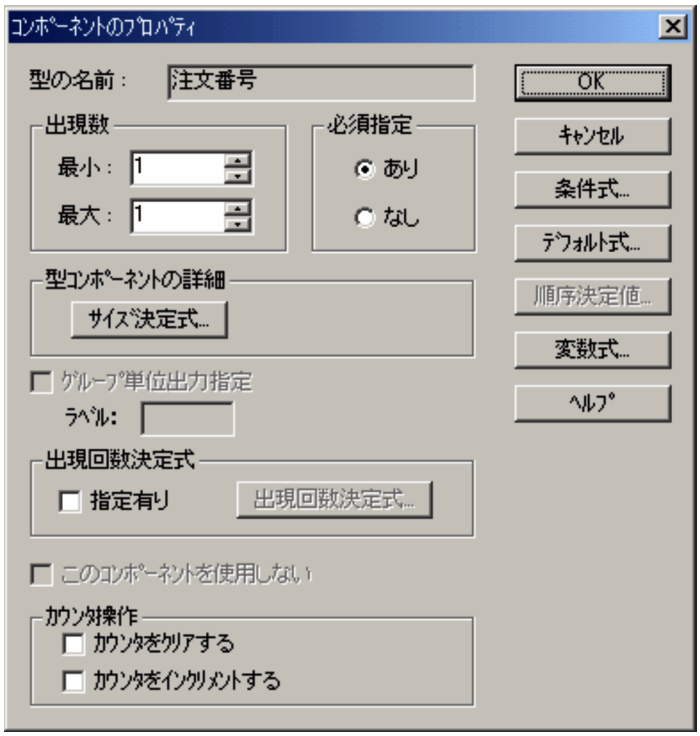
リストビューの要素を，ルート構造「MSG1」にドラッグ＆ドロップします。

4. **リストビューの「注文番号」を，ツリービューの構造「MSG1」にドラッグ＆ドロップします**



［コンポーネントのプロパティ］ダイアログが表示されます。

図 3-17 ［コンポーネントのプロパティ］ダイアログ



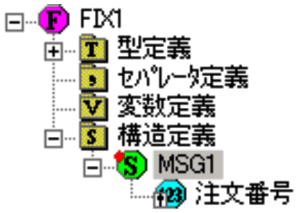
ダイアログ内の設定内容は次のとおりです。

- 出現数
  データ中にコンポーネントが繰り返し出現する回数を指定します。この構造では，コンポーネント「注文番号」の出現回数は 1 回なので，最小及び最大は初期値（「1」）のままにします。
- 必須指定
  MDL エディタでマッピングするときに，このコンポーネントのマップ式の定義を必須とするかどうかを指定します。マップ式定義は出力側コンポーネントに対して定義するため，フォーマットが入力専用の場合，指定しても機能しません。ここでは，入出力兼用フォーマットを定義するので「あり」を指定します。

上記以外は，初期値のままとします。

5. **［OK］ボタンをクリックします**

［コンポーネントのプロパティ］ダイアログを閉じて，構造「MSG1」アイテムの下にコンポーネントのアイテム「注文番号」が作成されます。

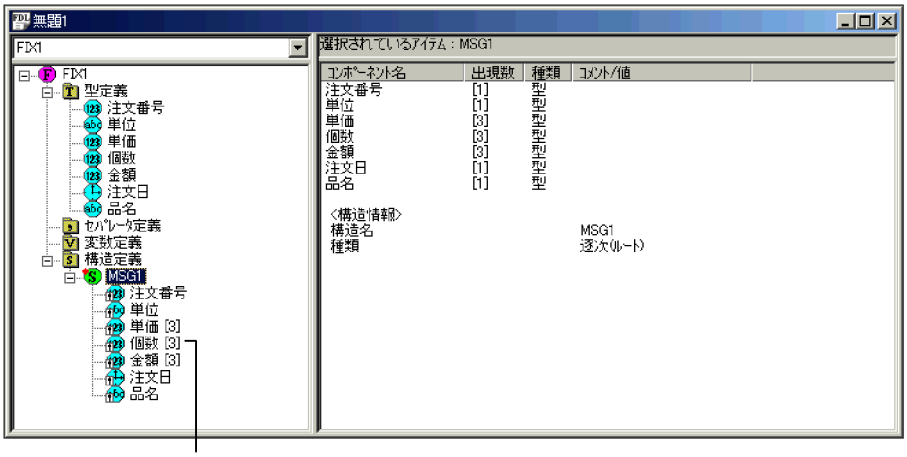


「注文番号」以外の要素も，同様に構造「MSG1」アイテムの下にドラッグ & ドロップして定義します。各コンポーネントの定義内容を表 3-3 に示します。

表 3-3　構造の定義

コンポーネント名	出現数		必須指定
	最小	最大	
注文番号	1	1	あり
単位	1	1	
単価	3	3	
個数	3	3	
金額	3	3	
注文日	1	1	
品名	1	1	

以上でフォーマット「FIX1」の構造が定義できました。



なお，型又は構造コンポーネントを子コンポーネントとして挿入する場合，親となるコンポーネントを選択した状態でドラッグ＆ドロップしたときは，その親の最後の子コンポーネントとして挿入されます。兄弟となるコンポーネントを選択した状態でドラッグ

＆ドロップしたときは，その兄弟コンポーネントの一つ前に挿入されます。

3.2.4 フォーマットの検証

フォーマットを MDL エディタに読み込んでマッピングする前に，FDL エディタでチェックコマンドを実行し，フォーマットにエラーがないことを確認する必要があります。

1. **［ツール］－［チェック］を選択します**
   アクティブになっているフォーマットの定義内容の正当性を検証します。検証結果は，［チェック結果］ダイアログに一覧表示されます。

図 3-18 ［チェック結果］ダイアログ

エラーがある場合，一覧でエラー内容をダブルクリック又は［ジャンプ］ボタンをクリックすると，エラーのあるデータがツリービュー内で選択状態になります。エラーを修正したら，再度検証を実行し，エラーがないことを確認してください。

2. **エラーがなければ，［閉じる］ボタンをクリックして［チェック結果］ダイアログを閉じます**

3.2.5 FDL ファイルの保存

フォーマットの定義内容にエラーがないことを確認したら，FDL に名前を付けて保存します。

1. **［ファイル］－［名前を付けて保存］を選択します**
   ［名前を付けて保存］ダイアログが表示されるので，ファイル名を指定します。
2. **［OK］ボタンをクリックします**
   以上で，FDL ファイルが作成できました。

3.3 複雑なデータ構造のフォーマット定義例

この節では，複雑なデータ構造のフォーマットを作成するときの，構造や式の定義について説明します。説明する内容を次に示します。

● 固定長形式のフォーマット定義

● レングスタグ形式のフォーマット定義

● セパレータ形式のフォーマット定義

● 条件式の定義

● デフォルト式の定義

● メッセージデータの例

各内容について次に説明します。

3.3.1 固定長形式フォーマットの定義

「3.2 簡単なデータ構造のフォーマット定義例」では，簡単な構造を持つ固定長形式のフォーマットを定義しました。ここでは少し複雑な構造を持つ固定長形式のフォーマット定義について説明します。説明するのは次の固定長フォーマットです。

- 繰り返し構造を持つメッセージ
- 複数種類のメッセージ

次に，各固定長形式のフォーマットの定義について説明します。

(1) 繰り返し構造を持つメッセージのフォーマット（フォーマット「FIX2」）

ここでは，繰り返し構造を持つメッセージのフォーマットを定義します。複数でまとまった意味を持つ要素で構成された構造が，繰り返し現れます。また，繰り返し現れる構造の下にも構造がネストしています。メッセージのデータ構造を図 3-19 に示します。

図 3-19　メッセージのデータ構造

メッセージのイメージ

注文番号　00001
注文日　981030
単位　PC(個)

No	明細			分納	
	単価(¥)	個数	金額(¥)	納入数	納入番号
1	1000	100	100000	50	000B001
				50	0000013
2	2000	200	200000	50	000B002
				150	0000014
3	10000	50	500000	30	000B003
				20	0000015

注文総額　¥ 800000

データフォーマットのイメージ

明細
分納

注文番号	単位	単価	個数	金額	納入数	納入番号	分納	明細	明細	注文日	注文総額
整数 5	文字列 1	整数 5	整数 5	整数 10	整数 5	文字列 7	12	44	44	日付 6	整数 15

（凡例）　数値はバイト数です。

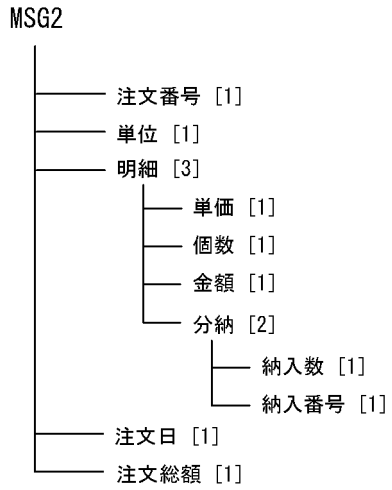
「単価」「個数」「金額」は三つの要素でまとまって意味を持ち，「分納」と同様に，構造「明細」の子コンポーネントです。構造「明細」の子コンポーネントである「分納」は，1 回のデータ中に 2 度出現します。そして，「分納」は「納入数」「納入番号」というコンポーネントの親の構造でもあります。

このメッセージデータのフォーマットを定義します。フォーマット名は，「FIX2」とします。ここでは，フォーマットの定義や型の定義についての説明を省略し，構造の定義について説明します。

(a) 構造の定義

構造を作成し，構造の下にコンポーネントをドラッグ＆ドロップし，構造を作成します。フォーマット「FIX2」の構造を図 3-20 に示します。

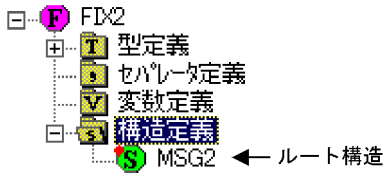
図 3-20 フォーマット「FIX2」の構造



（凡例） ［ ］内は出現回数です。

フォーマット「FIX2」の構造を定義します。

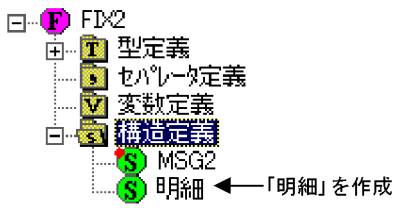
1. **ルート構造「MSG2」を作成します**



ルート構造「MSG2」は，「注文番号」「単位」「注文日」「注文総額」という四つの要素と，要素「単価」「個数」「金額」「分納」を持つ構造「明細」で構成されています。

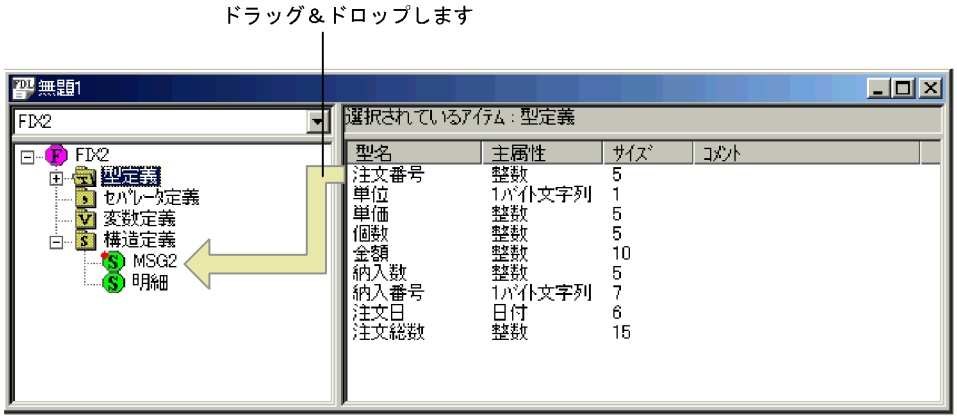
2. **ルート構造の下の構造「明細」を作成します**

ルート構造「MSG2」下に構造「明細」を作成します。既にルート構造は定義してあるので，作成される構造「明細」は普通の構造になります。構造「明細」下のコンポーネントは，定義した順序で出現する逐次構造です。構造「明細」には特別な条件はないので，［構造のプロパティ］ダイアログ以外の設定は必要ありません。

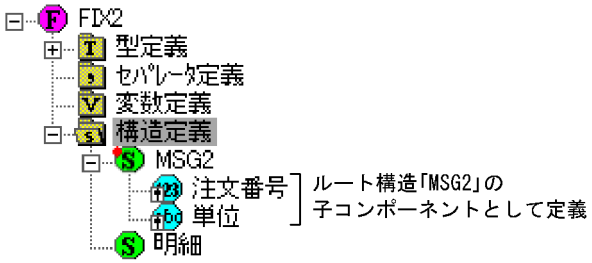


次に，ルート構造「MSG2」の子コンポーネントを定義します。

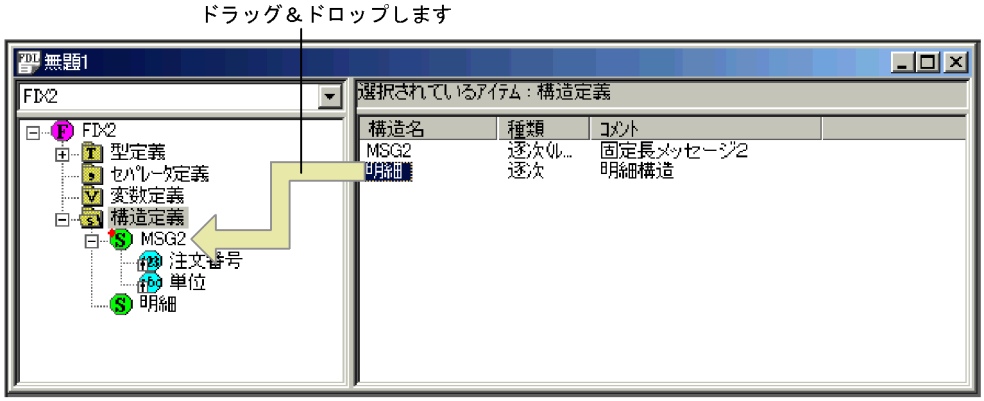
3. **要素「注文番号」をルート構造「MSG2」にドラッグ＆ドロップします**
   出現回数は，1 を設定します。
   型定義フォルダアイテムを選択し，リストビューからツリービューにデータをドラッグ＆ドロップすると容易に定義できます。



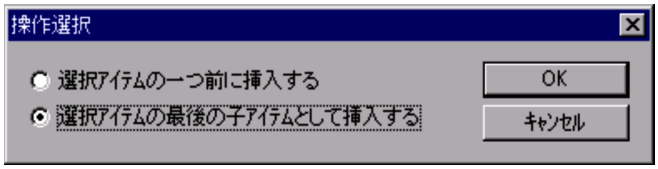
4. **要素「単位」をルート構造「MSG2」にドラッグ＆ドロップします**
   出現回数は，1 を設定します。



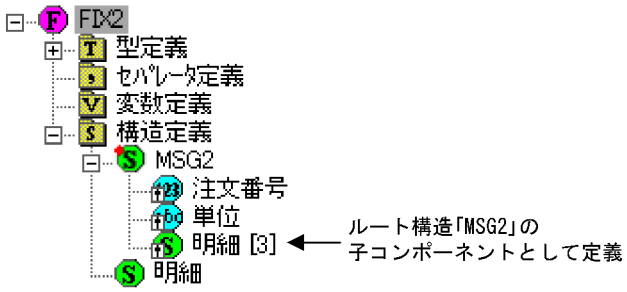
5. **作成した構造「明細」をルート構造「MSG2」にドラッグ＆ドロップします**
   構造定義フォルダアイテムを選択し，リストビューからツリービューにデータをドラッグ＆ドロップすると容易に定義できます。



ドラッグ＆ドロップしたいアイテムの挿入位置を問い合わせる［操作選択］ダイアログが表示されます。



ここでは，構造「明細」をルート構造「MSG2」の子コンポーネントとして挿入したいので，「選択アイテムの最後の子アイテムとして挿入する」を選択します。また，出現回数は，3 を設定します。



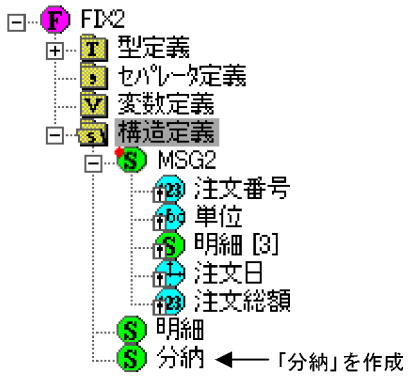
FDL ファイルでは，定義した構造やコンポーネントをルート構造の下にドラッグ＆ドロップし，データの構造をツリーで作成します。ただし，構造の実体は，構造定義フォルダの直下にあります。構造の下の構造は，実体の構造を参照する構造（サブコンポーネントと呼びます）になります。例えば，構造「明細」はルート構造「MSG2」の下のサブコンポーネントです。

ルート構造の下の構造「明細」は，構造定義フォルダの下の実体の構造「明細」を参照しているサブコンポーネントです。コンポーネント間は，自コンポーネントの上位構造から下位構造までを数箇所で参照できます。
次に，ルート構造「MSG2」のほかの子コンポーネントを定義します。

6. **要素「注文日」「注文総額」をルート構造「MSG2」にドラッグ＆ドロップします**
出現回数は，すべて 1 を設定します。

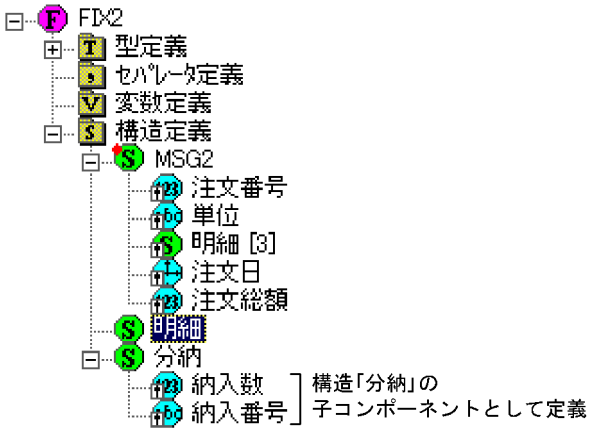
以上でルート構造の子コンポーネントの定義が完成しました。
次に，構造「明細」の子コンポーネントを定義します。構造「明細」の下には三つの要素と，「納入数」と「納入番号」を構造化した構造があります。まず，子の構造である構造「分納」を作成します。

7. **構造「分納」を作成します**



構造の種類は，逐次構造とします。
次に，構造「分納」のコンポーネントを定義します。

8. **「納入数」と「納入番号」を型定義からドラッグ＆ドロップします**
出現回数は，すべて 1 を設定します。



次に，構造「明細」の子コンポーネントを定義します。

9. **要素「単価」「個数」「金額」を構造「明細」にドラッグ＆ドロップします**
出現回数は，すべて 1 を設定します。
最後に，構造「分納」を構造「明細」の子コンポーネントとして定義します。

10. **構造「明細」の下に構造「分納」をドラッグ＆ドロップします**
出現回数は，2 を設定します。
これでフォーマット「FIX2」の定義が完成しました。

コンポーネントの作成順序には特に規定はありません。あらかじめ構造をすべて作成してから，最後に全体を組み立ててもかまいません。ただし，最終的にできたコンポーネントの順序が，実際のデータの順序と一致していることを確認してください。実際のデータの順序と一致していない場合は，ドラッグ＆ドロップでコンポーネントの順序を変更してください。

なお，サブコンポーネントには，直接ドラッグ＆ドロップして子コンポーネントを挿入できません。構造定義フォルダの直下にある実体の構造コンポーネントに子コンポーネントをドラッグ＆ドロップして挿入する必要があります。

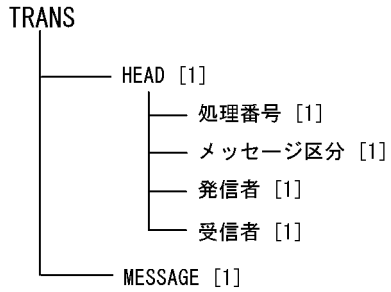
次に，複数のメッセージデータが変換できるフォーマットの定義について説明します。

(2) 複数種類のメッセージを持つ固定長形式のフォーマット（フォーマット「FIX」）

これまでの例題のメッセージのフォーマット定義例では，1 回の変換で 1 メッセージ中の 1 データしか変換できませんでした。ここでは，複数のメッセージデータが変換できるフォーマットの定義について説明します。

一般的に EDI データのメッセージには，そのメッセージの意味を示すヘッダ情報が付けられます。あるメッセージに対して，ヘッダ情報として処理番号，メッセージの種類を表すメッセージ区分，発信者，受信者情報を追加します。ヘッダ情報を追加したメッセージの構造を図 3-21 に示します。

図 3-21 複数種類のメッセージを持つフォーマットの構造



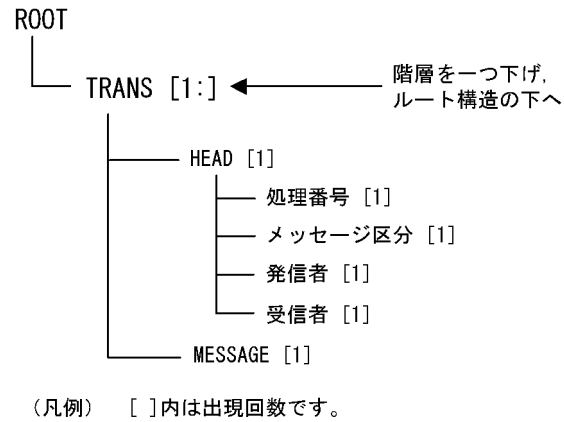
構造「MESSAGE」は，仮に配置したメッセージ用の構造で，メッセージ部分のルート構造に相当します。

ヘッダ情報を追加したメッセージの構造を使用して，複数種類のメッセージを持つフォーマットの定義について説明します。

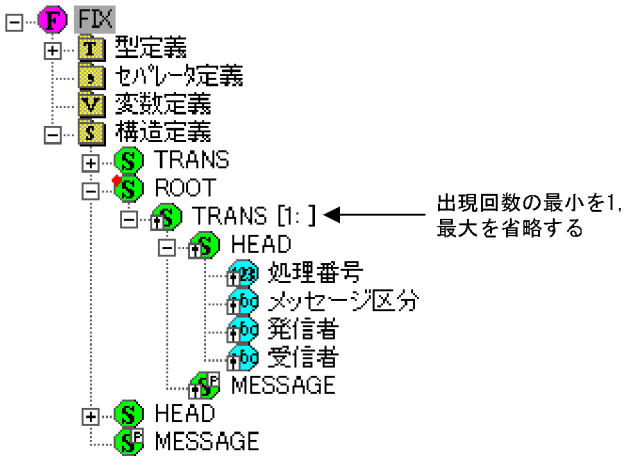
(a) 複数データを変換するフォーマットの定義

ヘッダ情報が付けられたメッセージが複数連続しているデータのフォーマットを定義します。データの構造を図 3-22 に示します。

図 3-22 複数データを変換するフォーマットの構造



ヘッダ情報を追加したメッセージが複数連続しているデータを変換するには，構造「TRANS」の階層を一つ下げて，ルート構造の下へ置きます。同時に，構造「TRANS」の出現回数は最小を 1 とし，最大を省略します。最大出現回数を省略すると出現回数は n 回となるので，データ数が増えてもデータを変換できるようになります。最大出現回数を省略する構造の定義は，その構造をルート構造の直下に置いて，1 単位の変換が n 回実行される構造を定義する以外には使用しないようにしてください。

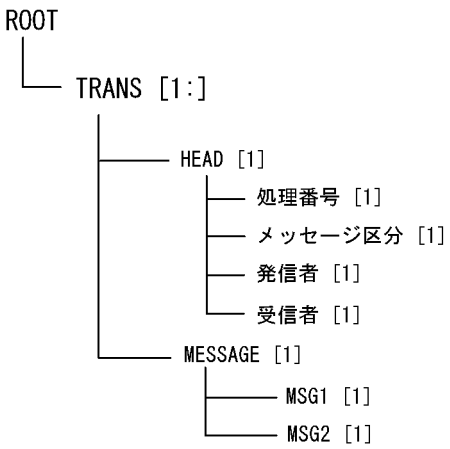


複数のメッセージを一つのフォーマットの中で定義し，データの値によって対応するメッセージ構造を選択するフォーマットの定義について説明します。

(b) 複数種類のメッセージを変換するフォーマットの定義（選択構造の定義）

構造中にメッセージ構造が二つあり，データの値によって，対応するメッセージ構造を選択してデータを変換するフォーマットを作成します。データの構造を図 3-23 に示します。フォーマット名は，「FIX」とします。

図 3-23　フォーマット「FIX」の構造



（凡例）　[ ]内は出現回数です。

構造「MESSAGE」下の二つのメッセージには，フォーマット「FIX1」で定義したメッセージ「MSG1」，フォーマット「FIX2」で定義したメッセージ「MSG2」を使用します。「MSG1」「MSG2」のデータ構造を図 3-24 に示します。

図 3-24 「MSG1」「MSG2」のデータ構造

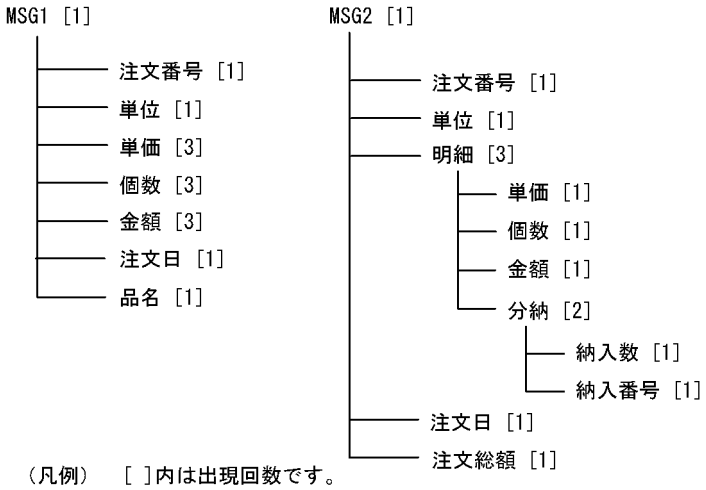


図 3-24 の定義について次に説明します。

● データをコピーする場合
コンポーネント「MSG1」「MSG2」は，各フォーマットからドラッグ＆ドロップでコピーできます。

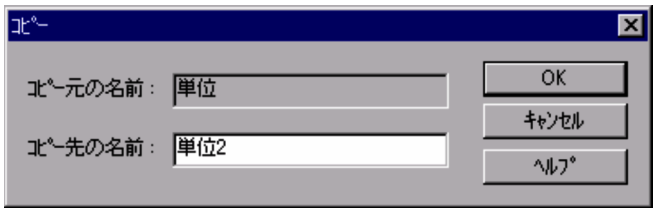
まず，「FIX1」から「MSG1」をコピーします。

1. **型定義フォルダ内の要素の型をコピーします**
   「FIX1」の要素の型をすべてコピーします。

2. **構造定義フォルダ内の構造をコピーします**
   「FIX1」から「MSG1」をコピーします。
   次に，「FIX2」から「MSG2」をコピーします。

3. **型定義フォルダ内の要素の型をコピーします**
   「FIX1」と同一名で定義の同じ要素の型はコピーしません。
   「FIX2」だけで定義されている要素「納入数」「納入番号」「注文総額」をコピーします。また，要素「単位」は，「FIX1」の要素「単位」とは異なる値の定義を持つ型であるという想定でコピーします。ただし，同一名で異なる定義はできないので，「MSG2」の要素「単位」には異なる名前を付ける必要があります。「MSG1」の要素「単価」をコピーした後に，「MSG2」の要素「単位」をコピーしようとすると，コピー先の名前を指定するダイアログが表示されます。

図 3-25 ［コピー］ダイアログ



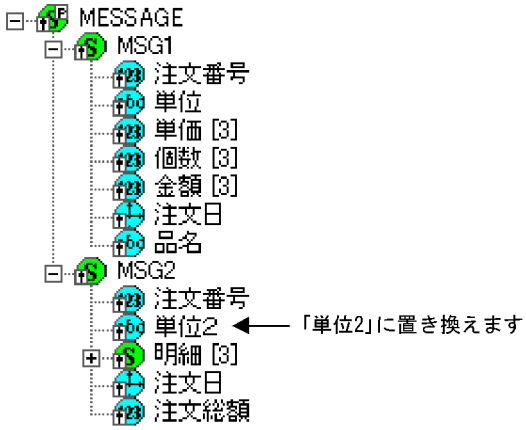
この例では，「MSG2」の要素「単位」には，新しい要素名「単位 2」を付けてコピーします。

4. **構造定義フォルダ内の構造をコピーします**

「MSG2」をコピーします。コンポーネントとして使用されている「分納」「明細」も同時にコピーされます。

「MSG2」の要素「単位」に新しい要素名「単位 2」を付けたことに伴い，「MSG2」の子コンポーネント「単位」もコンポーネント「単位 2」とする必要があります。

「MSG2」の子コンポーネント「単位」を削除してから，型定義フォルダから「単位 2」を「MSG2」の下にドラッグ＆ドロップし，コンポーネントを「単位 2」に置き換えてください。



以上でデータがコピーできました。

● 選択構造の定義

構造「MESSAGE」下のコンポーネント「MSG1」「MSG2」は，変換するデータの値によってどちらかのメッセージが選択されるように定義します。この場合，構造「MESSAGE」を選択構造として定義します。選択構造にすると，子コンポーネントとして定義したコンポーネントのどれかのデータがデータ中に出現するように定義できます。選択構造の子コンポーネントを定義する場合，最小の出現回数は 1 回以上に設定する必要があります。この例では，メッセージ「MSG1」「MSG2」の出現回数はどちらも 1 回固定を設定します。

次に，選択構造「MESSAGE」下のコンポーネント「MSG1」「MSG2」のうち，どちらかが出現できるように定義します。

● 出現するコンポーネントの選択条件の定義

トランスレータは，変換するデータの値が選択構造の子コンポーネントのうちどれと一致するかを判定します。選択構造だけを宣言した場合，コンポーネントが定義された順に，変換するデータとコンポーネントが適合するかどうかを評価し，最初に正当であると評価されたものを選択します。

この評価順序は，順序決定式を定義して変更できます。親の選択構造のコンポーネントには評価順序の条件を表す式（順序決定式）を記述し，子のコンポーネントには自コンポーネントを選択する場合の値（順序決定値）を指定します。このように定義しておくと，データを変換するときに式を評価し，同じ値を持つコンポーネントを優先的に評価します。順序決定式の詳細については，「6.6.3　順序決定式」を参照してください。

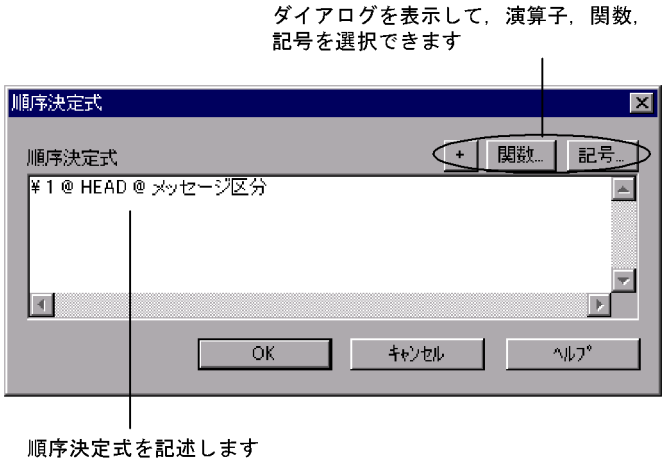
この例では，ヘッダ情報の中に，データがどのメッセージのものであるかを示すメッセージ区分があります。このメッセージ区分を使用して，選択構造の子コンポーネントが選択できるように定義します。

まず，選択構造「MESSAGE」に対して，メッセージを選択するための順序決定式を記述します。順序決定式は，［順序決定式］ダイアログに記述します。

1. **構造「MESSAGE」の［構造のプロパティ］ダイアログで［順序決定式］ボタンをクリックします**

［順序決定式］ダイアログが表示されます。

図 3-26　［順序決定式］ダイアログ



2. **順序決定式を記述します**

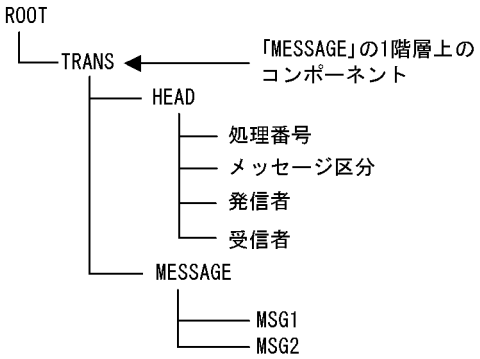
ここでは，メッセージ区分の値によって振り分けるようにするので，メッセージ区分のコンポーネント名を記述します。コンポーネント名は，フォーマット名，ルート構造から対象コンポーネントまでを「@」でつないだもので指定します。

```
FIX@TRANS@HEAD@メッセージ区分
```

略記法を使用して上記の式を記述すると，次のようになります。

```
¥1@HEAD@メッセージ区分
```

「¥n」は n 階層上のコンポーネントを意味する略記法です。「MESSAGE」の直接の親，つまり 1 階層上のコンポーネント「TRANS」を指定したいので「¥1」と記述します。



構造「TRANS」の下の構造「HEAD」の子コンポーネントである「メッセージ区分」の値によって，順序を決定することになります。
次に，コンポーネントに対して順序決定式に対応する値を定義します。コンポーネント「MSG1」「MSG2」に対して，メッセージ区分の値を設定します。
順序決定値は，［順序決定値］ダイアログに記述します。

3. **各コンポーネントの［コンポーネントのプロパティ］ダイアログで［順序決定値］ボタンをクリックします**
［順序決定値］ダイアログが表示されます。

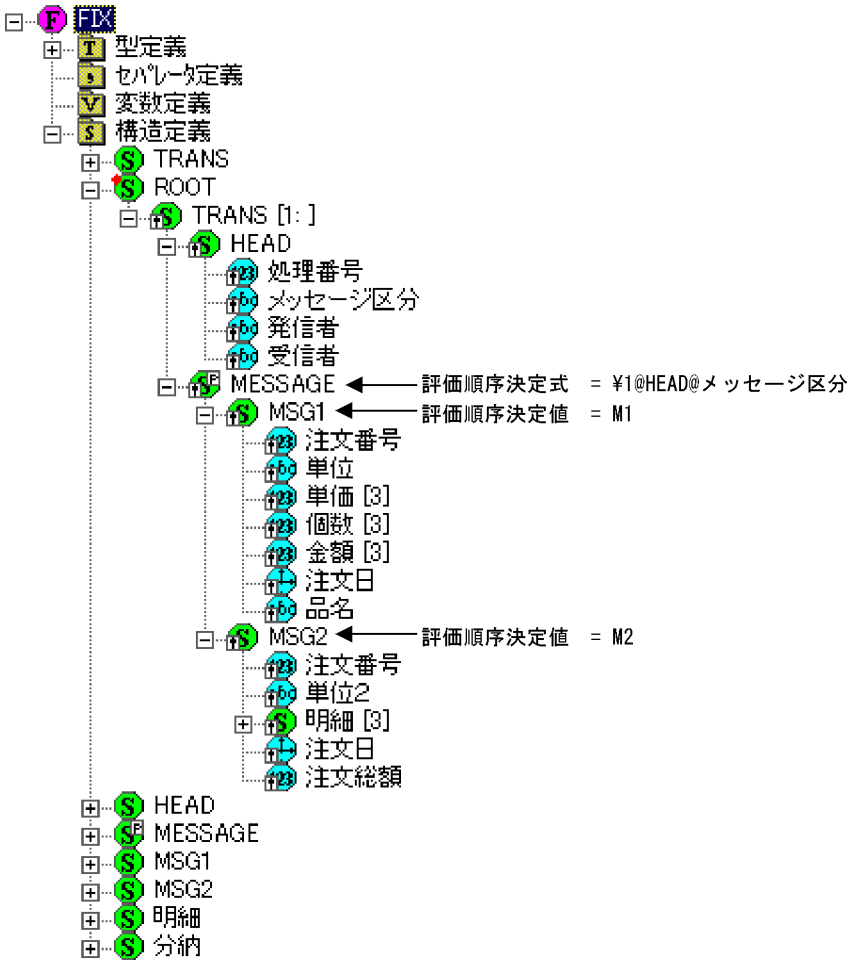
4. **順序決定式に対応する値を記述します**

図 3-27 ［順序決定値］ダイアログ

順序決定式に対応する値と値の属性を設定します。選択構造の選択条件の値は，親の

選択構造の順序決定式で記述した内容と同じ型の値を設定する必要があります。
この例では，「MSG1」に対しては「M1」，「MSG2」に対しては「M2」という，メッセージ区分の値を設定します。要素「メッセージ区分」と同じ型「文字列２バイト」として記述するので，値の属性に「文字列」を指定します。メッセージ区分の値が「M1」のときはメッセージ「MSG1」が，値が「M2」のときはメッセージ「MSG2」が選択されます。
これで，変換するデータの値によって，選択構造「MESSAGE」の子コンポーネント「MSG1」又は「MSG2」が選択できるようになりました。

以上で，複数種類のメッセージを持ち，複数データを変換できる固定長データのフォーマット「FIX」が定義できました。



次に，レングスタグ形式のフォーマットの定義について説明します。

3.3.2 レングスタグ形式フォーマットの定義

データの形式には，各要素のサイズと値を定義した形式があります。データサイズを表す要素があるので，一般的に「レングスタグ形式」と呼びます。レングスタグ形式は，CII 標準で使用される形式です。

ここでは，これまでの例で定義した固定長フォーマットを使用して，レングスタグ形式のフォーマット定義について説明します。データ全体は，ヘッダとメッセージの繰り返しから構成されますが，ヘッダ部分は固定長フォーマットのままで，メッセージ本体の部分をレングスタグ方式としたフォーマットを定義します。

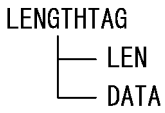
ここでは，次の例を使用してレングスタグ形式のフォーマット定義について説明します。

- 一つの要素に対するレングスタグ構造のフォーマット
- 固定長形式のメッセージ「MSG1」のレングスタグ構造化
- 固定長形式のメッセージ「MSG2」のレングスタグ構造化
- 複数種類のメッセージを持つレングスタグ形式のフォーマット

次に，各レングスタグ形式フォーマットの定義について説明します。

(1) 一つの要素に対するレングスタグ構造のフォーマット

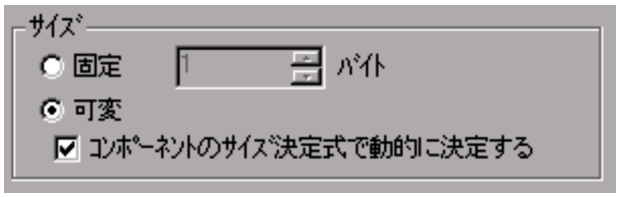
ここでは，一つの要素に対するレングスタグ構造のフォーマット定義について説明します。レングスタグは，データのサイズを表す要素とデータの実体である要素から構成されます。



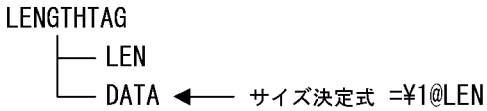
「LEN」がサイズを表す要素，「DATA」が要素データの実体です。各要素の構造定義にこの構造を使用し，レングスタグ構造のフォーマットを作成します。

レングスタグの構造では，構造のデータが入力データの場合と，出力データの場合とではサイズの定義方法がやや異なります。各定義方法について次に説明します。

● レングスタグ構造のデータを入力データとする場合
要素「DATA」のサイズは要素「LEN」が持っているため，要素のサイズが動的に決定されることになります。要素「DATA」に対しては，［型の詳細］ダイアログのサイズ定義で，要素のサイズは「可変」,「コンポーネントのサイズ決定式で動的に決定する」と指定します。



また，コンポーネント「DATA」には，サイズ決定式で，コンポーネント「LEN」によってサイズが決定することを記述します。



サイズ決定式は，［サイズ決定式］ダイアログで記述します。［サイズ決定式］ダイアログは，サイズ決定式を定義するコンポーネントの［コンポーネントのプロパティ］ダイアログで［サイズ決定式］ボタンをクリックすると表示されます。
サイズ決定式には要素のサイズを持つコンポーネント名を記述します。

図 3-28 ［サイズ決定式］ダイアログ

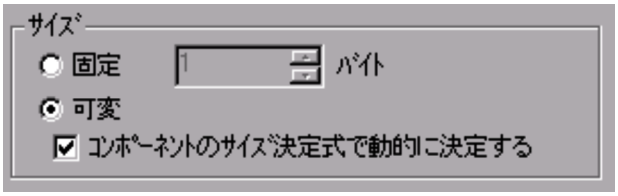
ここでは，コンポーネント「DATA」のサイズはコンポーネント「LEN」によって動的に決定されることになるので，サイズ決定式には「LEN」のコンポーネント名を記述します。コンポーネント名は，フォーマット名，ルート構造から対象コンポーネントまでを「@」でつないだもので指定します。ここでは，n 階層上のコンポーネントを意味する略記法「¥n」を使用して次のように記述します。

```
¥1@LEN
```

「DATA」の 1 階層上のコンポーネント「LENGTHTAG」を指定したいので「¥1」と記述しています。これで「LENGTHTAG」の子コンポーネント「LEN」によって「DATA」のサイズが決定されます。

なお，一つのレングスタグ構造を複数箇所で使用する場合，要素サイズを持つコンポーネントのグローバル名は一意に決定できません。サイズ決定式は，略記法を使用して相対的な名前で記述するようにしてください。
動的にサイズを決定する指定と，コンポーネントのサイズ決定式は対で指定する必要があります。
サイズ決定式の詳細については，「6.6.2　サイズ決定式」を参照してください。

● レングスタグ構造のデータを出力データとする場合
要素「DATA」のサイズは，マップ式によって代入される値などによって決まります。



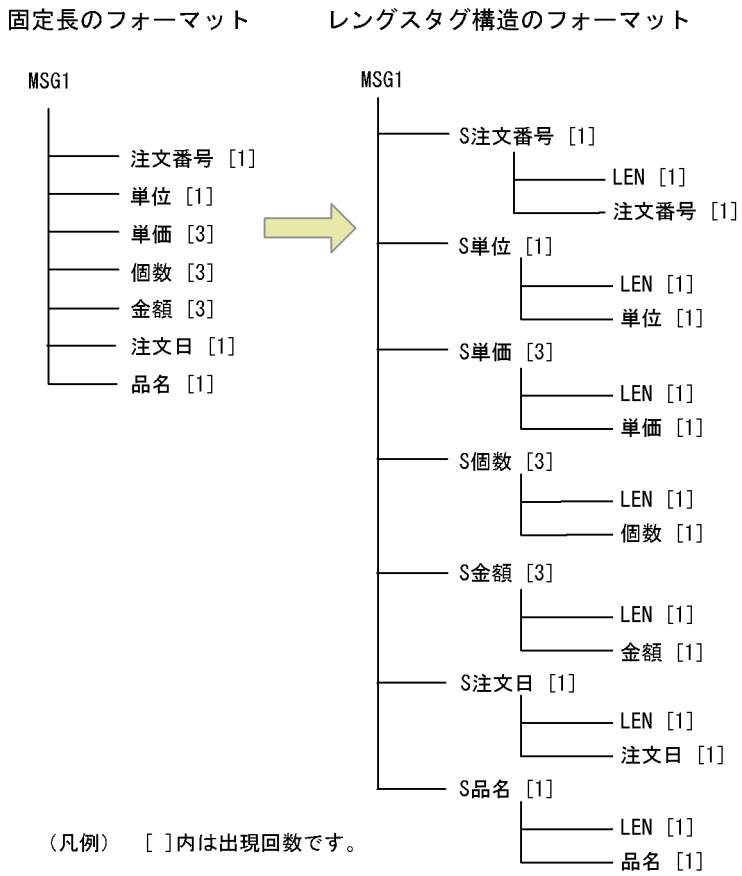
サイズ定義で「可変」,「コンポーネントのサイズ決定式で動的に決定する」と指定します。なお，サイズ決定式は定義しないでかまいません。要素サイズが固定長の場合は，レングスタグ構造であっても，要素のサイズ定義は「固定」でバイト数を指定するだけです。

要素「LEN」に対しては，特別な定義は必要ないので普通に型を定義します。ここでは，データ種別に「数値，符号無 2 進整数」，サイズを「2」バイトとします。2 進整数を使用する場合は，フォーマットのプロパティで定義するエンディアンに注意してください。

次に，これまで定義した固定長フォーマットの例を使用して，レングスタグ形式のフォーマットの定義について説明します。

（2）固定長形式のメッセージ「MSG1」のレングスタグ構造化（フォーマット「LT1」）

ここでは，固定長フォーマット「FIX1」で定義したメッセージ「MSG1」をレングスタグ構造にしたフォーマットを定義します。



データの項目はフォーマット「FIX1」と同様ですが，個々の要素データの前にサイズを表す要素を挿入し，フォーマット「FIX1」で要素だったものは，この例では構造となります。フォーマット名は，「LT1」とします。

(a) フォーマットの定義

フォーマットを作成し，プロパティを定義します。固定長フォーマット「FIX1」の場合と同様に，不要文字削除指定のチェックを入れ，文字列の後続スペースや数値の前後の 0 などの余分な文字を削除します。レングスタグ構造のサイズを表す要素「LEN」には 2 進数を使用するので，ここではビッグエンディアンを指定します。

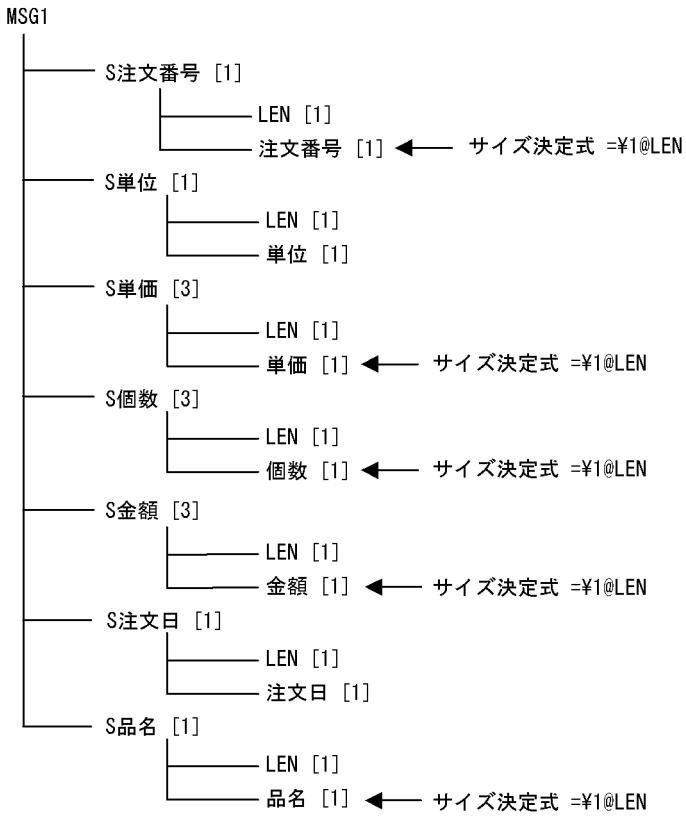
(b) 型の定義

サイズの定義以外は，固定長フォーマット「FIX1」と同様に定義します。要素のサイズ定義は，「単位」「注文日」はサイズを可変とする必要がないため固定長のままとします。ほかの要素のサイズ定義は「1 バイト以上の可変」，「コンポーネントのサイズ決定式で動的に決定する」と指定します。

(c) 構造の定義

「LEN」以外の各要素に対してレングスタグ構造を定義し，ルート構造の子コンポーネントとします。この例では，レングスタグ構造の名前は，各要素名に「S」を付けたものとします。フォーマット「LT1」の構造を図 3-29 に示します。

図 3-29 フォーマット「LT1」の構造



（凡例） [ ]内は出現回数です。

「単位」と「注文日」はレングスタグ構造化しますが，固定長のためサイズ決定式は記述しません。

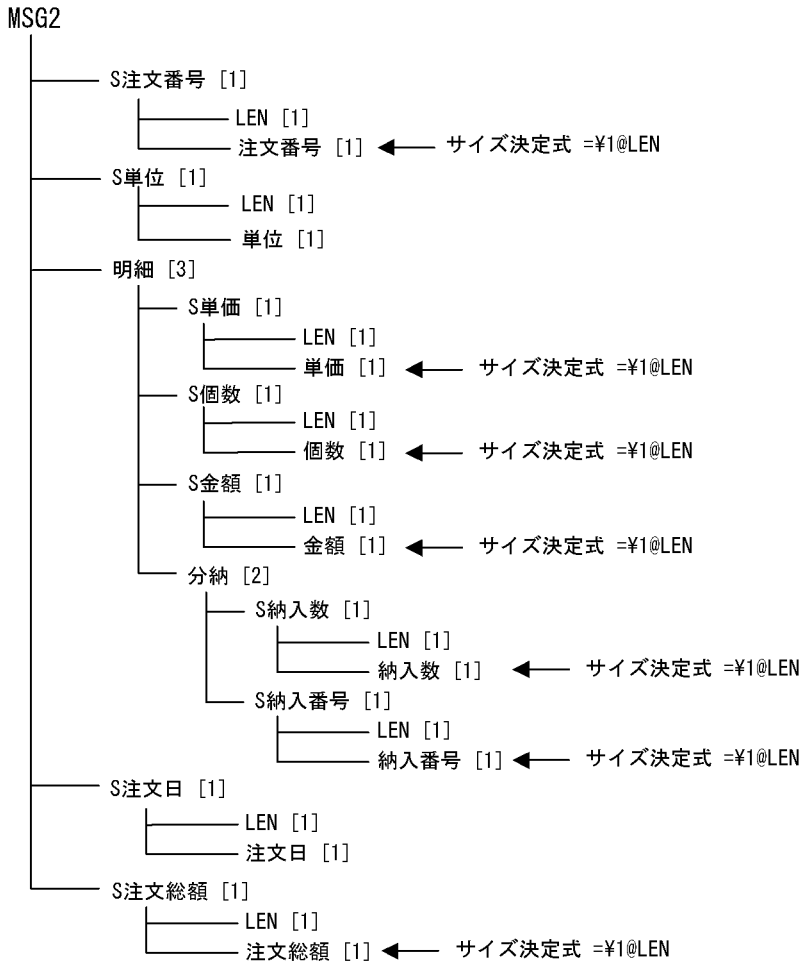
以上で，フォーマット「FIX1」のメッセージ「MSG1」をレングスタグ構造化したフォーマット「LT1」の定義が完成しました。

(3) 固定長形式のメッセージ「MSG2」のレングスタグ構造化（フォーマット「LT2」）

繰り返し構造を持つ固定長フォーマット「FIX2」のメッセージ「MSG2」をレングスタグ構造にしたフォーマットを定義します。レングスタグ構造化の方法は，「MSG1」の場

合と同様です。要素「単位」と「注文日」を固定長，残りの要素はすべて可変長として定義し，構造を作成します。データの構造を図 3-30 に示します。フォーマット名は，「LT2」とします。

図 3-30　フォーマット「LT2」の構造



（凡例）［ ］内は出現回数です。

(4) 複数種類のメッセージを持つレングスタグ形式のフォーマット（フォーマット「LT」）

ここでは，フォーマット「LT1」「LT2」で作成したレングスタグ形式のメッセージフォーマットにヘッダ部を追加し，一つのフォーマットにまとめます。複数データ及び複数メッセージに対応したレングスタグ形式のフォーマットを定義します。フォーマット名は，「LT」とします。

ヘッダ部は固定長のままで，構造「TRANS」「HEAD」「MESSAGE」はフォーマット「FIX」と同様に定義します。構造「MESSAGE」の子コンポーネント「MSG1」にはフォーマット「LT1」の構造「MSG1」を，「MSG2」にはフォーマット「LT2」の構造「MSG2」を使用します。「MSG1」及び「MSG2」は，各フォーマットから要素及び構造をドラッグ＆ドロップでコピーして使用できます。

以上でフォーマット「LT」の定義が完成しました。

3.3.3 セパレータ形式フォーマットの定義

セパレータ形式のフォーマットとは，個々の要素データをセパレータで区切った形式のフォーマットです。代表的なものに CSV（Comma Separated Value）があります。CSV とは，データベースやスプレッドシートのデータをコンマ「,」で区切ってテキストファイル形式にしたものです。EDI 標準の中では UN/EDIFACT がセパレータ形式のデータを扱っています。

固定長フォーマットのデータでは個々の要素サイズが決まっているため，どこまでが一つの要素のサイズか分かります。また，レングスタグでは要素のデータサイズはデータごとに異なりますが，サイズを表す要素があるのでデータのサイズが分かります。これに対して，サイズが規定されない可変長要素では，その要素の終わりを示すためにセパ

レータが必要になります。

ここでは，次の例を使用してセパレータ形式のフォーマット定義について説明します。

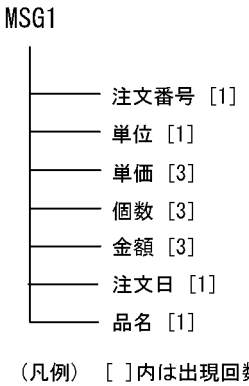
- 1 種類のセパレータを使用したフォーマット
- 複数のセパレータと解放文字を使用したフォーマット
- 複数種類のメッセージを持つセパレータ形式のフォーマット

次に，各セパレータ形式フォーマットの定義について説明します。

(1) 1 種類のセパレータを使用したフォーマット（フォーマット「SEPA1」）

固定長フォーマット「FIX1」のメッセージ「MSG1」にセパレータを使用したフォーマットを定義します。データの構造を図 3-31 に示します。

図 3-31 「MSG1」の構造



メッセージ「MSG1」は，7 種類の要素が順番に合計 13 個現れます。要素は階層化されていないため，単純な区切り文字を使用します。フォーマット名は，「SEPA1」とします。

1 種類のセパレータを使用したフォーマット「SEPA1」を定義します。

(a) フォーマット名の定義

レングスタグ構造の場合と同様に，不要文字削除の指定はチェックを入れます。また，この例では，要素のサイズのカウント方法に「文字数」を指定します。文字数で数えた場合，2 バイト文字の 1 文字のサイズは「1」として数えます。また，文字列中に解放文字があっても，文字数としては数えません。

(b) 型の定義

要素「単位」「注文日」を固定長，ほかの要素は可変長とします。セパレータを使用するフォーマットでは，レングスタグ構造の定義と異なり，「コンポーネントのサイズ決定式で動的に決定する」と指定する必要はありません。

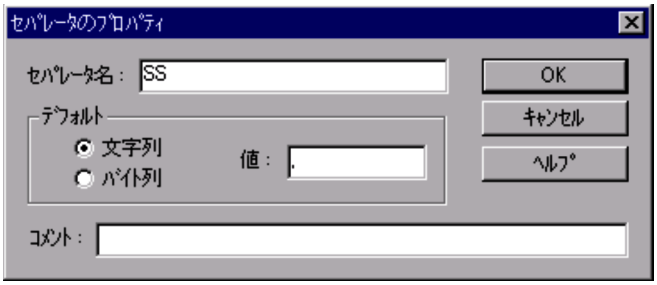
(c) セパレータの定義

セパレータの名前や値を定義します。セパレータを定義しておくと，セパレータを使用する構造は，セパレータの名前を指定するだけで，値を記述する必要はありません。複数の構造で同じ値をセパレータとして使用する場合などは，セパレータを定義しておくと便利です。セパレータの値を文字列で定義した場合，フォーマットのプロパティで定義したセパレータ文字コードが適用されます。

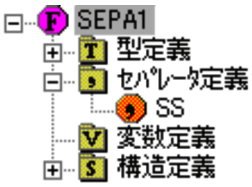
セパレータを新規作成します。

1. **［挿入］－［セパレータ］を選択します**
   ［セパレータのプロパティ］ダイアログが表示されます。

図 3-32 ［セパレータのプロパティ］ダイアログ



2. **セパレータ名と値を指定します**
   この例では，セパレータ名として「SS」，その値としてコンマ「,」を定義します。
3. **［OK］ボタンをクリックします**
   セパレータ定義フォルダアイテムの下にセパレータ「SS」が作成されます。



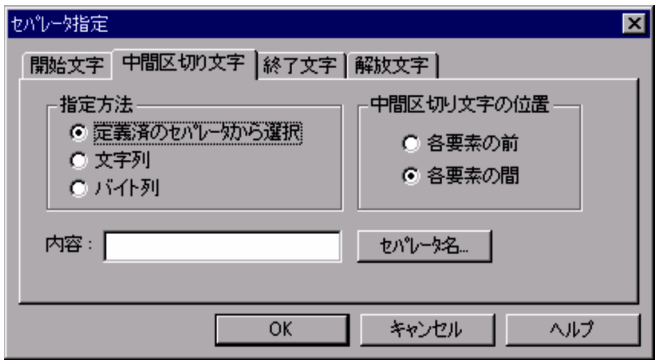
(d) 構造の定義

固定長フォーマット「FIX1」と同様にツリー構造を定義します。次に，構造に使用するセパレータの設定について説明します。

この例では，ルート構造「MSG1」に対して，セパレータ「SS」の値「,」を設定します。

1. **ルート構造「MSG1」の［構造のプロパティ］ダイアログで［セパレータ］ボタンをクリックします**
   ［セパレータ指定］ダイアログが表示されます。

図 3-33 ［セパレータ指定］ダイアログ



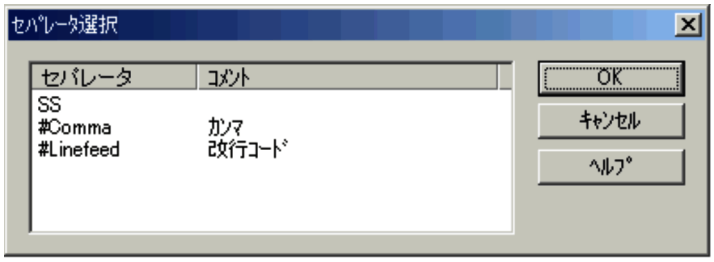
セパレータの種類やセパレータの指定方法などを指定します。この例では，要素間を区切るセパレータなので，「中間区切り文字」として指定します。
ダイアログの設定内容は次のとおりです。

- 指定方法
  使用するセパレータを定義しているので，「定義済のセパレータから選択」を指定します。
- 中間区切り文字の位置
  要素間を区切るセパレータを設定したいため，「各要素の間」を指定します。

次に，定義済のセパレータを指定します。

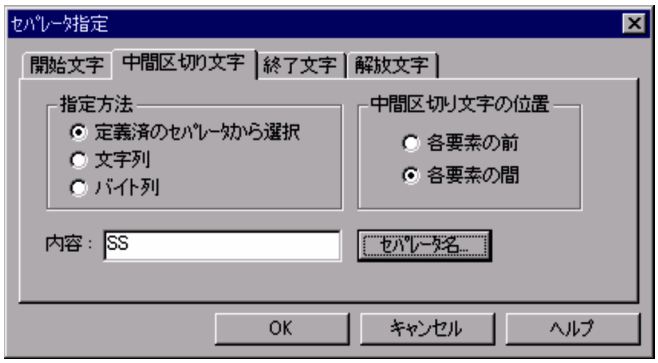
2. **［セパレータ名］ボタンをクリックします**
   ［セパレータ選択］ダイアログが表示されます。

図 3-34 ［セパレータ選択］ダイアログ



ダイアログの一覧には，選択している構造で指定できるセパレータ名が表示されます。

3. **一覧からセパレータ名「SS」を選択して，［OK］ボタンをクリックします**
   ［セパレータ指定］ダイアログの内容入力欄に，選択したセパレータ名「SS」が入力されます。



セパレータの設定が完成しました。

以上でフォーマット「SEPA1」の定義が完成しました。

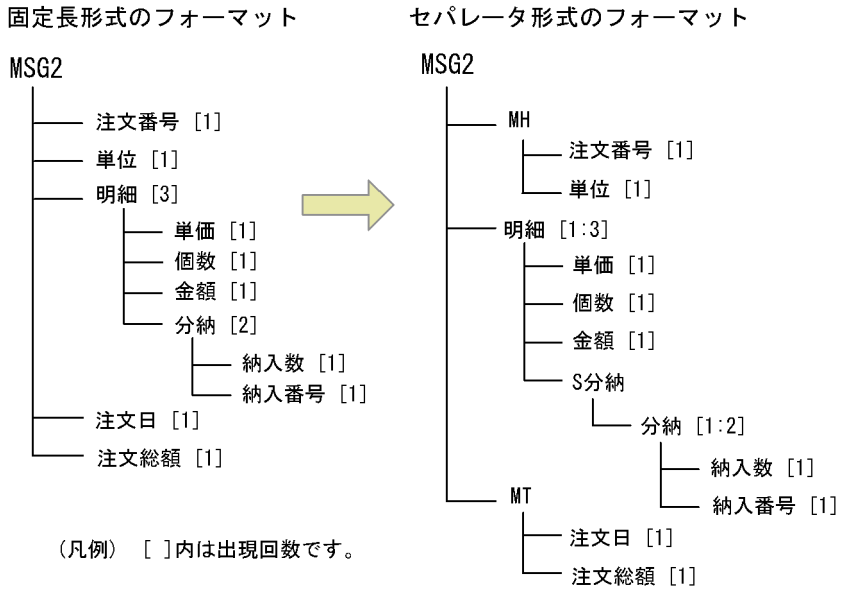
メッセージ「MSG1」では，要素中の普通のデータにセパレータと同じ文字を使用した場合，その文字がセパレータか，要素のデータの一部かどうかを区別する解放文字を使用していません。したがって，各要素中にセパレータと同じ文字「,」がないようにしてください。「,」があった場合は，要素中のデータの一部である「,」の位置で，要素が終わったと認識されてしまうため，正しくデータを変換できなくなります。

次に，複数のセパレータと解放文字を使用したフォーマットの定義について説明します。

(2) 複数のセパレータと解放文字を使用したフォーマット（フォーマット「SEPA2」）

固定長形式のフォーマット「FIX2」のメッセージ「MSG2」に複数のセパレータと解放文字を使用したフォーマットを定義します。固定長形式とセパレータ形式のフォーマットの比較を図 3-35 に示します。

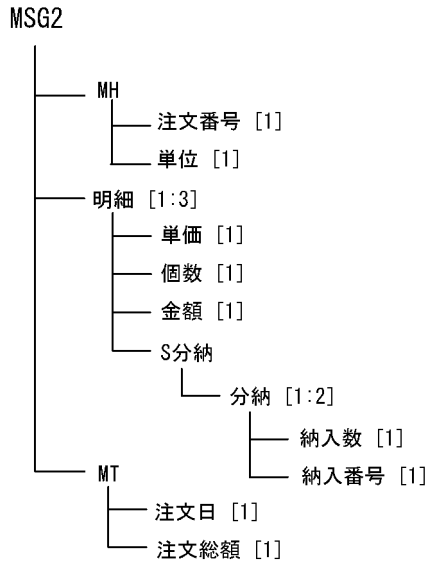
図 3-35　固定長形式とセパレータ形式のフォーマットの比較



「単価」「個数」「金額」は三つの要素でまとまって意味を持ち，「分納」と同様に，構造「明細」の子コンポーネントです。構造「明細」の子コンポーネントである「分納」は，1 回のデータ中に最大 2 回出現します。そして，「分納」は「納入数」「納入番号」というコンポーネントの親の構造でもあります。

固定長やレングスタグ構造のメッセージでは固定回出現だった構造「明細」「分納」を，可変回出現できるように変更します。データの構造を図 3-36 に示します。フォーマット名は，「SEPA2」とします。

図 3-36　フォーマット「SEPA2」の構造



（凡例）［ ］内は出現回数です。

セパレータ形式のフォーマットでは，メッセージを 3 分割します。メッセージの先頭情報として「注文番号」と「単位」，メッセージ本体の情報として「明細構造」，終了情報として「注文日」と「注文総額」に分けます。さらに，構造「分納」を可変回出現できるようにするために，構造を 1 階層増やします。これは，構造「分納」の出現回数をセパレータで分かるようにするためです。

メッセージ「MSG2」に複数のセパレータと解放文字を使用したフォーマットを定義します。

(a) フォーマット及び型の定義

フォーマット「SEPA1」の例と同様です。要素「単位」「注文日」を固定長，ほかの要素は可変長とします。

(b) セパレータ名の定義

「MSG2」の子コンポーネントは，セパレータとして，開始文字「タグ名」，終了文字「改行」，各要素の前には中間区切り文字「+」を使用します。

構造「明細」の子である構造「分納」は，複数回出現するコンポーネントなので，構造「分納」間の区切り文字として「*」，構造「分納」の子コンポーネントの区切りには「:」を使用します。

また，文字列中にセパレータと同じ文字を使用できるようにするため，解放文字として「？」を使用します。セパレータは個々のコンポーネントのプロパティで使用する値などを定義しますが，解放文字はルート構造に対して指定します。

● セパレータ及び解放文字の定義
タグ名とするセパレータを除く 4 種類のセパレータと解放文字を定義します。各セパレータ及び解放文字の定義内容を表 3-4 に示します。

表 3-4　セパレータと解放文字の定義内容

名前	種別／値	備考
TS	バイト列／ 0x0d0a	構造「MSG2」の子コンポーネントの終了文字
RS	文字列／ *	複数回出現する構造「分納」の中間区切り文字
CS	文字列／ :	構造「分納」の子コンポーネント間の中間区切り文字
ES	文字列／ +	構造「MSG2」の子コンポーネント間の中間区切り文字
RC	文字列／ ?	解放文字

セパレータ及び解放文字は，セパレータ文字コードの文字列又はバイト列として定義できます。終了文字としての改行は，バイト列として定義します（0x0D0A）。その他は文字列で定義します。
次に，セパレータの使用例について説明します。

● セパレータの使用例
これらのセパレータを使用したデータは次のような形式になります。タグ名としては「MH」,「MD」,「MT」を使用します。ここでは，実際のデータ値の代わりに要素名を記述してあります。

```
MH+注文番号+単位（改行）
MD+単価+個数+金額+納入数:納入番号*納入数:納入番号（改行）
MD+単価+個数+金額+納入数:納入番号*納入数:納入番号（改行）
MD+単価+個数+金額+納入数:納入番号*納入数:納入番号（改行）
MT+注文日+注文総額（改行）
```

構造「明細」は可変回数出現するので,「MD」で始まるデータは 1 行分だけでもかまいません。また，構造「分納」も 1 回又は 2 回の可変回出現として定義すれば，構造「分納」の 2 回目がないデータは次のような形式になります。

```
MD+単価+個数+金額+納入数:納入番号（改行）
```

次に，解放文字の使用例について説明します。

● 解放文字の使用例
「MSG2」の要素の中間区切り文字として，セパレータ「+」を使用しているため，要素のデータとして「+」を使用する場合は，解放文字「?」を使用します。例えば，
データの一部として「A ＋ B」というデータがある場合は,「A?+B」としてセパレータと区別する必要があります。このフォーマットでは要素サイズのカウント方法に文

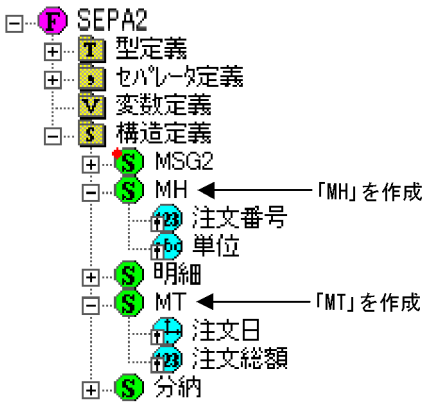
字列を指定しているので，「A?+B」として解放文字を挿入した場合でも文字列のサイズは「3」のままです（要素サイズのカウント方法にバイト数を指定した場合は 4 になります）。

次に，構造を定義します。

(c) 構造の定義

まず，「MSG2」の子コンポーネントを 3 分割します。

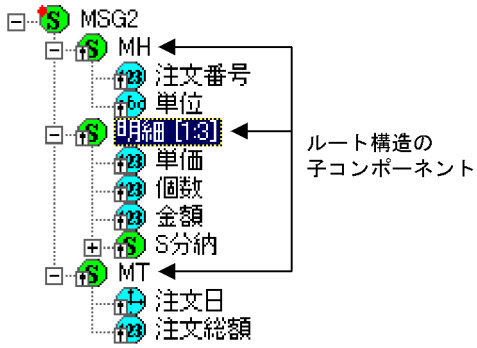
1. **「注文番号」と「単位」を子コンポーネントとする構造「MH」，「注文日」と「注文総額」を子コンポーネントとする構造「MT」を作成します**



2. **構造「明細」の子コンポーネントとして構造「S 分納」を作成します**
   出現回数は 1 とします。
3. **構造「分納」を構造「S 分納」の子コンポーネントとして定義します**
   出現回数は最小 1，最大 2 とします。

4. **ルート構造「MSG2」の下には子コンポーネントとして，「MH」「明細」「MT」を定義します**

   構造「明細」の出現回数は最小 1，最大 3 とします。



5. **構造で使用するセパレータを設定します**

   セパレータを設定する構造と設定内容を表 3-5 に示します。

表 3-5　セパレータの設定

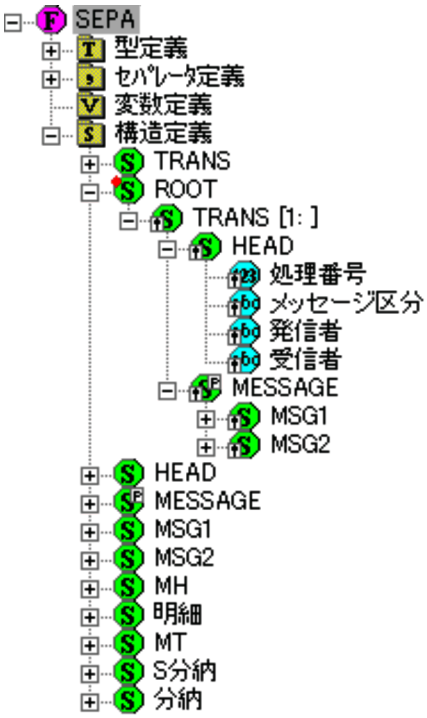
構造名	セパレータ名／値	セパレータの種類	備考
MSG2	RC ／ ?	解放文字	—
MH	MH	開始文字	指定方法「文字列」
	ES ／ +	中間区切り文字	中間区切り文字の位置「各要素の前」

構造名	セパレータ名／値	セパレータの種類	備考
	TS ／ 0x0d0a	終了文字	－
明細	MD	開始文字	指定方法「文字列」
	ES ／ +	中間区切り文字	中間区切り文字の位置「各要素の前」
	TS ／ 0x0d0a	終了文字	－
S 分納	RS ／ *	中間区切り文字	中間区切り文字の位置「各要素の間」
分納	CS ／ :	中間区切り文字	中間区切り文字の位置「各要素の間」
MT	MT	開始文字	指定方法「文字列」
	ES ／ +	中間区切り文字	中間区切り文字の位置「各要素の前」
	TS ／ 0x0d0a	終了文字	－

以上で複数セパレータを使用したメッセージフォーマット「SEPA2」の定義が完成しました。

（3） 複数種類のメッセージを持つセパレータ形式のフォーマット（フォーマット「SEPA」）

ここでは，フォーマット「SEPA1」と「SEPA2」で作成したセパレータ形式のメッセージフォーマットにヘッダ部を追加し，一つのフォーマットにまとめます。複数データ及び複数メッセージに対応したセパレータ形式のフォーマットを定義します。フォーマット名は，「SEPA」とします。



ヘッダ部は，メッセージ「SEPA2」と同様にセパレータを使用した構造にします。構造「MESSAGE」の子コンポーネント「MSG1」にはフォーマット「SEPA1」の構造「MSG1」を，「MSG2」にはフォーマット「SEPA2」の構造「MSG2」を使用します。「MSG1」及び「MSG2」は，各フォーマットから要素及び構造をドラッグ＆ドロップでコピーして使用できます。

この場合のデータ形式は次のような形式になります。

```
MD+処理番号+ メッセージ区分 + 発信者+受信者 （改行）
```

メッセージ本体を選択構造化する方法は，これまでの例と同様です。ただし，解放文字の使用を宣言するコンポーネントはルート構造である，ということに注意してください。
フォーマット「SEPA2」では，解放文字は構造「MSG2」に対して定義しました。
フォーマット「SEPA」では，解放文字は構造「ROOT」に対して定義し，「MSG2」に対する解放文字の定義を削除します。

以上でフォーマット「SEPA」の定義が完成しました。

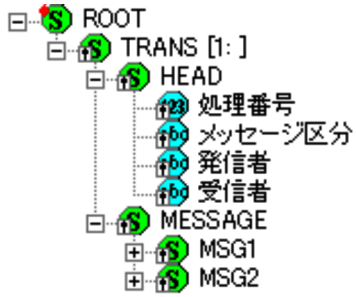
3.3.4 条件式の定義

これまでの例では，データの構造を定義してきました。FDL エディタでは，データの構

造を定義する以外に，データなどに対する条件を設定できます。データに対する条件を定義する式を「条件式」と呼びます。

フォーマット「FIX」を例にして，条件式について説明します。フォーマット「FIX」の構造を図 3-37 に示します。

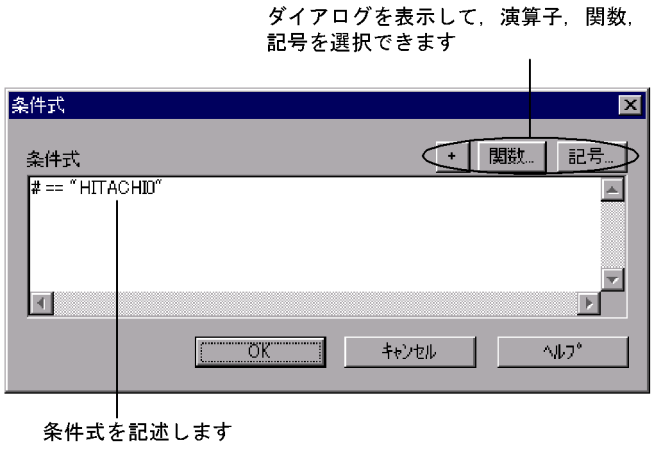
図 3-37　フォーマット「FIX」の構造



例えば，ヘッダ部の受信者の値が「HITACHI0」以外の場合は，データ不正としてエラー処理することとします。このような条件は，コンポーネントの条件として記述できます。

1. **コンポーネント「受信者」の［コンポーネントのプロパティ］ダイアログで［条件式］ボタンをクリックします**
   ［条件式］ダイアログが表示されます。

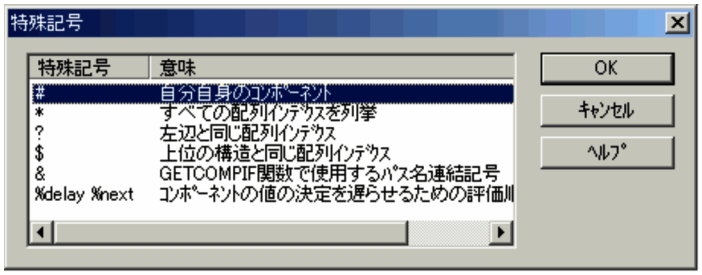
図 3-38　［条件式］ダイアログ



2. **コンポーネントに対する条件式を記述します**
   ここでは，コンポーネント「受信者」の値は「HITACHI0」である，という条件を定義します。条件式を定義するコンポーネント自身を指定する場合は，通常，自コンポーネントを表す略記法として記号「#」を使用します。記号は直接入力したり，ダイアログから選択したりできます。ここでは，ダイアログから選択します。

3. **［記号］ボタンをクリックします**
［特殊記号］ダイアログが表示されます。

図 3-39 ［特殊記号］ダイアログ



条件式に使用する記号を選択し，［OK］ボタンをクリックします。ここでは，自コンポーネントを表す「#」を選択します。選択した記号が条件式に挿入されます。
自コンポーネント「受信者」の値は「HITACHI0」である，という条件を定義します。文字列定数はダブルクォーテーションで囲んでください。条件式を次に示します。

```
# == "HITACHI0"
```

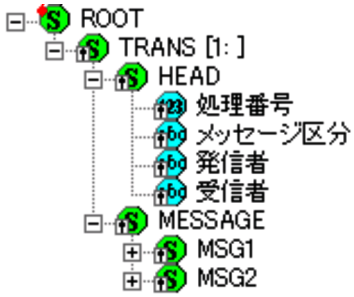
この場合，フォーマットの実際のデータで受信者の値が「HITACHI0」以外の場合は，トランスレータは評価規則が不正であるとして，エラー処理します。以上で条件式が定義できました。

3.3.5 デフォルト式の定義

FDL エディタではマップ式は記述しませんが，特定の要素に対してあらかじめ決まっている値を「デフォルト値」として定義できます。MDL エディタでは，この要素の値がデフォルト値のままで問題なければ，マップ式を記述して要素の値を定義する必要はありません。

フォーマット「FIX」を例にして，デフォルト値の設定について説明します。フォーマット「FIX」の構造を図 3-40 に示します。

図 3-40 フォーマット「FIX」の構造



ヘッダ部の発信者に対して，デフォルト値として「HITACHI1」を定義します。

1. **コンポーネント「発信者」の［コンポーネントのプロパティ］ダイアログで［デフォルト式］ボタンをクリックします**
   ［デフォルト式］ダイアログが表示されます。

図 3-41 ［デフォルト式］ダイアログ

2. **コンポーネントに対するデフォルト式を記述します**
   デフォルト式に記述できるのは，定数の代入式だけです。マップ式と同様に，デフォルト式の後尾にセミコロンを付けてください。ここでは，コンポーネント「発信者」のデフォルト値を「HITACHI1」と定義します。デフォルト式を次に示します。

```
= "HITACHI1" ;
```

   この場合，デフォルト式を定義したフォーマットを出力フォーマットとして使用したときは，MDL エディタでコンポーネント「発信者」に対してマップ式を記述しなければ，値として常に「HITACHI1」が出力されます。マップ式を記述した場合は，デフォルト式よりもマップ式の定義が優先されます。
   以上でデフォルト式が定義できました。

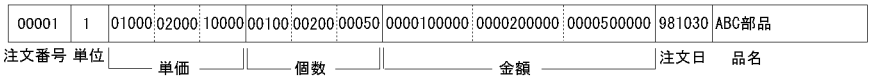
3.3.6 メッセージデータの例

ここでは，これまの例で定義してきた固定長，レングスタグ，セパレータの各形式のメッセージデータの例を示します。この例では，レングスタグ及びセパレータ形式のデータは，不要文字を削除しています。

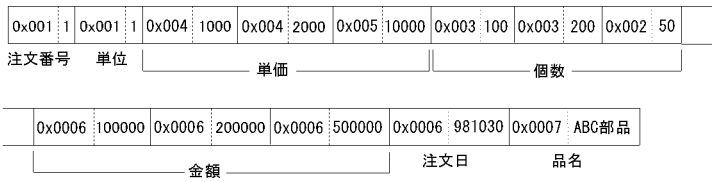
メッセージ「MSG1」のデータの例を図 3-42 に示します。

図 3-42 メッセージ「MSG1」のデータの例

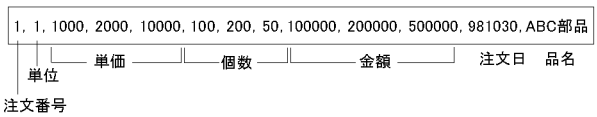
固定長形式



レングスタグ形式



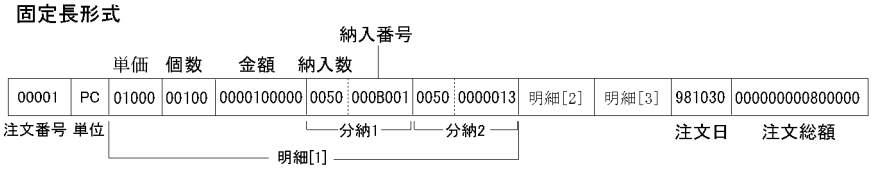
セパレータ形式



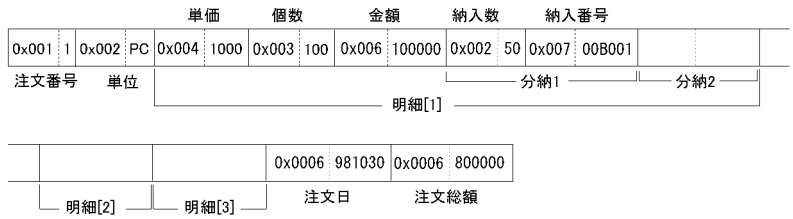
メッセージ「MSG2」のデータの例を図 3-43 に示します。

図 3-43　メッセージ「MSG2」のデータの例

固定長形式



レングスタグ形式



セパレータ形式

MH+1+PC	注文番号／単位
MD+1000+100+100000+50:000B001*50:0000013	明細 [1]
MD+2000+200+200000+50:000B002*150:0000014	明細 [2]
MD+10000+50+500000+30:000B003*20:0000015	明細 [3]
MT+981030+800000	注文日／注文総額

3.4 FDL ファイル作成時の注意事項

FDL ファイルのツリー構造は，データの各要素の出現順序を表します。データ構造の作成方法は様々に考えられるので，「3.2 簡単なデータ構造のフォーマット定義例」以降で説明した構造の作成例は一つの例に過ぎません。変換するデータと照らし合せて，要求を満たす構造を作成してください。なお，構造を作成するときには，次の点に注意してください。

(1) 要素の定義

要素の型を定義するときの注意事項は，属性ごとに三つに分類されます。

- 整数，実数，暗黙的小数部付数値要素の場合
- 日付要素の場合
- 文字列要素の場合

次に，要素を定義するときの注意事項について属性ごとに説明します。

(a) 要素の属性が整数，実数，暗黙的小数部付数値の場合

- 要素サイズが固定長の場合は，符号・小数点・指数記号を桁数として数えます。
- 要素サイズが可変長の場合は，符号・小数点・指数記号は桁数として数えません。また，要素の最大サイズは設定されません。
- 最大桁数を指定した場合は，整数部の前に 0 を付けたりして，指定桁数分の数字が出力されます。
- 有効数字だけを出力したい場合は，フォーマットのプロパティで，不要な文字を削除して出力するように指定してください。
- データ中で数値文字列内で使用されるような文字（例えば，「,」「+」など）をセパレータとして使用する場合，符号，同一文字の桁セパレータ及び指数記号は使用しないでください。また，CSV フォーマットの場合は，「"」も使用しないでください。

(b) 要素の属性が日付時刻の場合

YYMMDD 型は 6 桁の数値で表しますが，上位 2 桁は西暦年を意味するため，数値へ変換した場合の値としては 8 桁の数値になります。したがって，YYMMDD 型で読み込んだ要素の値をそのまま整数 6 桁の要素にマッピングすると，桁あふれが発生します。入力した日付型の要素の値を出力する場合は，出力側の要素も日付型を使用してください。

(c) 要素の属性が文字列の場合

文字列型の要素で，文字コードとして 2 バイト文字のコードを指定した場合は，埋め字は 2 バイト文字でなければなりません。埋め字をバイト列で指定する場合も 2 バイトの指定が必要です。2 バイト文字以外の文字コードでは，埋め字は 1 バイト（文字）で指定します。

(2) セパレータの定義

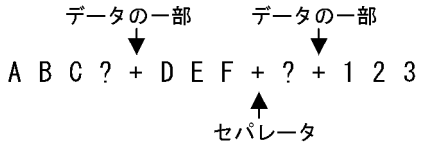
可変長の要素，可変回数出現するコンポーネントに対しては，原則的に要素の終わりを示したり，コンポーネントの出現の終了を示したりする，区切りとしてのセパレータが必要になります。セパレータを定義するときの注意事項について次に説明します。

- 可変長の要素コンポーネントの場合は，コンポーネント間の中間区切り文字や，終了文字のセパレータを定義します。
- 不特定回数出現するコンポーネントの場合は，次に現れるコンポーネントに開始文字のセパレータを定義します。ただし，そのコンポーネントが入出力データの最後尾にある場合，開始文字のセパレータは必要ありません。
- セパレータの値は，文字又は 16 進数で定義します。文字の場合は，フォーマット内で使用する初期値のセパレータ文字コードになります。16 進数で複数バイトの値を指定した場合は，その値はフォーマットのプロパティで指定したエンディアンに従う値として解釈されます。例えば，数値「1」を 2 バイトの 16 進数（2 バイト）で表す場合は，ビッグエンディアンでは「0x0001」，リトルエンディアンでは「0x0100」と記述します。

(3) 解放文字の定義

要素の文字列中で，セパレータと同じ文字を使用する場合は，その文字が，要素の終わりを示すセパレータなのか，要素のデータの一部なのかを区別するために，解放文字を使用する必要があります。解放文字の直後の文字は，セパレータではなく，要素のデータの一部として解釈されます。

次に，セパレータとして「+」，解放文字として「?」を使用した場合の入力データの例を示します。



解放文字「?」の直後の文字「+」は，要素のデータの一部として扱われるので，要素は「ABC+DEF」と「+123」の二つあることになります。

要素の文字列中で，解放文字と同じ文字を使用する場合は，セパレータと同じように，文字の直前に解放文字を指定する必要があります。解放文字の定義方法はセパレータと同じですが，解放文字はルート構造に設定します。設定した解放文字は，そのフォーマット全体に対して適用されます。

(4) 構造の定義

構造を定義する場合の注意事項は，次の 3 種類あります。

- コンポーネントの出現回数を省略する場合

- 選択構造を定義する場合
- 構造全体を定義する場合

次に，各内容について説明します。

(a) コンポーネントの出現回数を省略する場合

コンポーネントの最大出現回数を省略すると，マップ式の記述に制限が発生します。コンポーネントがまとまって意味を持つ構造（例えば，単独であってもデータとして完結している構造）以外には，極力最大出現回数を省略しないでください。

(b) 選択構造を定義する場合

選択構造を使用する場合，できる限りコンポーネントの評価順序を指定してください。評価順序の指定は，構造に対して順序決定式を，子コンポーネントに対して順序決定値を定義します。コンポーネントの評価順序を指定するには，評価順序を指定する選択構造が現れる前に，子コンポーネントを選択するための条件値を持つ要素が必要です。順序決定式の詳細については，「6.6.3　順序決定式」を参照してください。

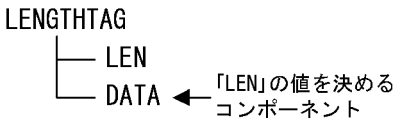
コンポーネントの評価順序が指定できない場合は，選択構造の子コンポーネントに対して，開始文字のセパレータを定義するか，又は正当性を示す条件式を定義して，コンポーネントを正しく選択できるようにしてください。また，選択構造の子コンポーネントの最小出現回数は 1 回以上を指定してください。

(c) 構造全体を定義する場合

フォーマットの構造は，入力側と出力側の階層の深さが，同程度になるように定義してください。階層の深さが著しく異なる場合は，マップ式が複雑になったり，記述できなくなる場合があります。

また，出力側のフォーマットで，レングスタグ構造のデータのように，先に出力されるコンポーネントの値が，後から出力されるコンポーネントによって決まる場合は，マップ式の評価を遅延させる指定でマップ式を定義する必要があります。

<出力側>



評価順序遅延を指定するコンポーネントのサイズは，固定長にしてください。評価順序遅延指定のマップ式の詳細については，「6.6.6(3)　評価順序遅延指定のマップ式」を参照してください。

4 MDL エディタの操作

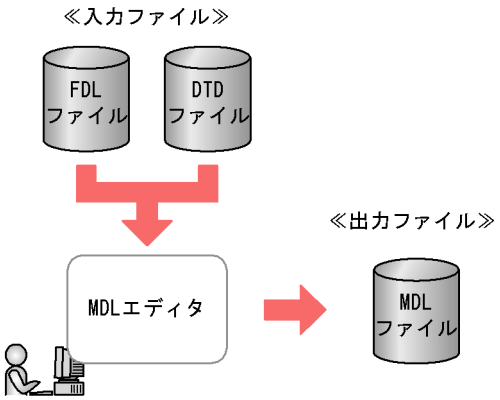
この章では，MDL エディタの操作の概要，具体例を使用した MDL エディタの操作方法について説明します。

4.1　操作の概要

この節では，MDL ファイルを作成するための MDL エディタの操作手順，MDL エディタの画面構成，MDL エディタのコマンドについて説明します。

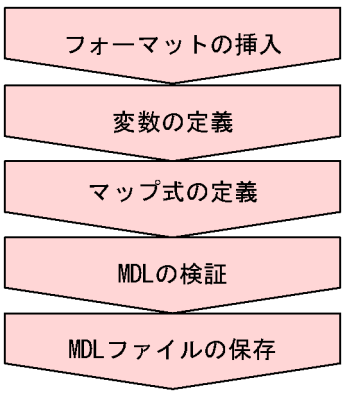
4.1.1　MDL ファイルの作成手順

MDL エディタでは，FDL エディタで定義した入力フォーマット，出力フォーマットのコンポーネント間を対応付けます。FDL ファイルからフォーマットを読み込んで，マッピングします。また，XML フォーマットを変換する MDL ファイルの場合は，FDL ファイルではなく DTD ファイルを読み込んで，MDL ファイルのフォーマットとしてマッピングします。フォーマット間のマッピング定義をファイルとして保存したものが MDL ファイルです。



MDL ファイルの作成手順を図 4-1 に示します。

図 4-1　MDL ファイルの作成



手順について次に説明します。

1. **フォーマットの挿入**
   マッピングする入力側，出力側のフォーマットを読み込みます。フォーマットは FDL エディタで作成した FDL ファイル，又は DTD ファイルから取り込みます。読み込んだフォーマットに対応する入力データのファイル名，出力データのファイル名を設定します。フォーマットは複数読み込むことができるので，複数のフォーマット間でもマッピングできます。

2. **変数の定義**
   変換時に使用する変数の名前や値を定義します。変数を使用しないデータ又は FDL エディタで設定済みの場合，変数を定義する必要はありません。

3. **マップ式の定義**
   入力側，出力側のフォーマットのコンポーネント間を対応付け，マップ式を定義します。

4. MDL **の検証**
   チェックコマンドでマッピングの定義内容の正当性を検証します。データを変換する前に，この検証を実行してエラーがないか確認する必要があります。

5. MDL **ファイルの保存**
   MDL に名前を付けて保存します。これで MDL ファイルが作成できます。MDL ファイルの拡張子は，「.mdl」です。

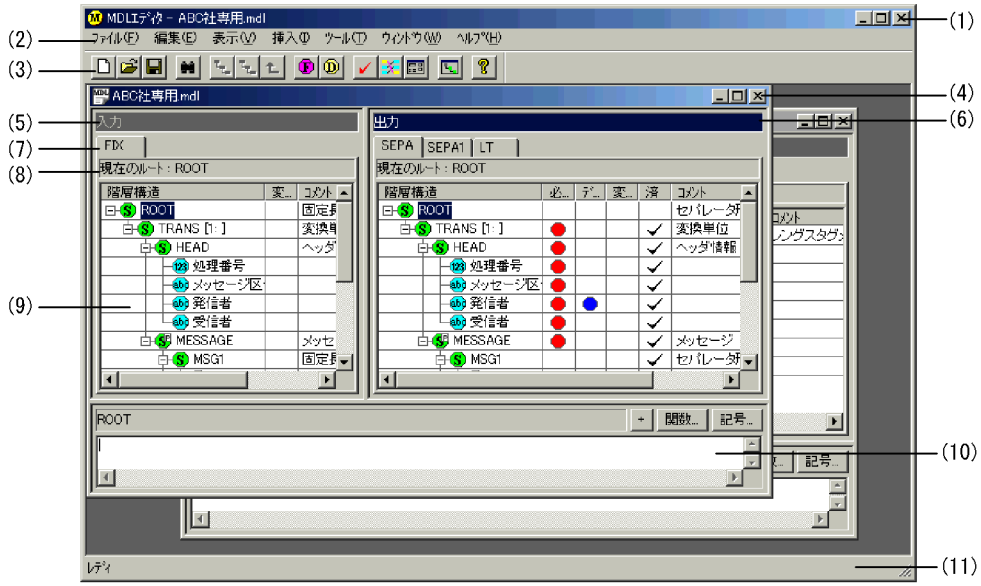
4.1.2 MDL エディタの画面構成

ここでは，MDL エディタの画面構成について説明します。MDL エディタは次の操作で起動できます。

1. **［スタート］－［プログラム］－［**Interschema**］－［**MDL **エディタ］を選択します**
   MDL エディタが起動します。MDL エディタは，同時に複数の MDL ファイルを開くことができます。

MDL エディタの画面構成図 4-2 に示します。

図 4-2　MDL エディタの画面構成



画面内の各部分について次に説明します。図 4-2 の番号は，説明の番号と一致しています。

(1) MDL エディタウィンドウ

MDL エディタのメインとなるフレームウィンドウです。

(2) メニューバー

実行するコマンドを指定するためのメニューです。コマンドの詳細については，MDL エディタのヘルプを参照してください。

(3) ツールバー

使用頻度の高いコマンドのボタンを集めたバーです。マウスでドラッグすると，任意の位置に移動できます。各ボタンの上にマウスカーソルを移動させると，ステータスバーにボタンの説明が表示されます。

(4) ドキュメントビューウィンドウ

一つのドキュメントに対して複数個表示できるウィンドウです。一度に複数のドキュメントを編集できます。ただし，MDL エディタで一度に編集できるドキュメントは 10 個までです。

(5) 入力フォーマットビュー

入力側のフォーマットを表示するビューです。

(6) 出力フォーマットビュー

出力側のフォーマットを表示するビューです。

(7) フォーマットセレクトタブ

複数のフォーマットの中から編集したいフォーマットを選択するタブです。タブをクリックすることで，編集したいフォーマットを切り換えられます。

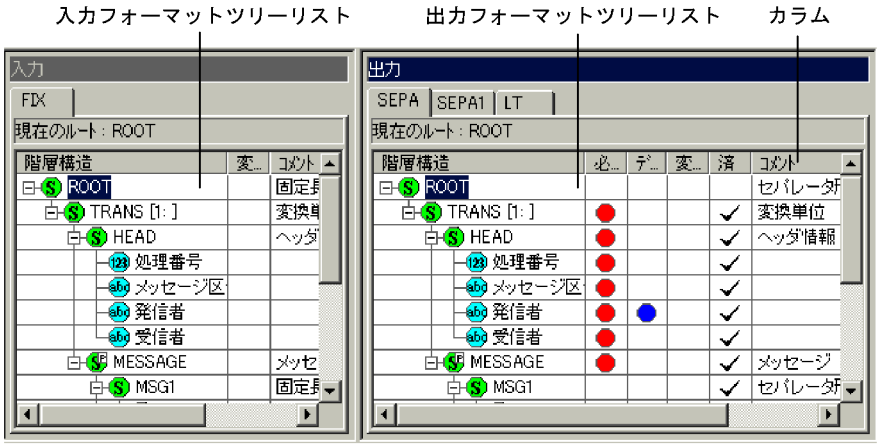
(8) ルートコンポーネント情報

現在，ツリーのルートとして表示されているコンポーネント名を表示するエリアです。

(9) フォーマットツリーリスト

フォーマット内のコンポーネントの階層構造，属性，マッピング状況などを表示するコントロールです。

図 4-3　フォーマットツリーリスト



フォーマットツリーリストのカラムについて，表 4-1 に説明します。

表 4-1　フォーマットツリーリストのカラム

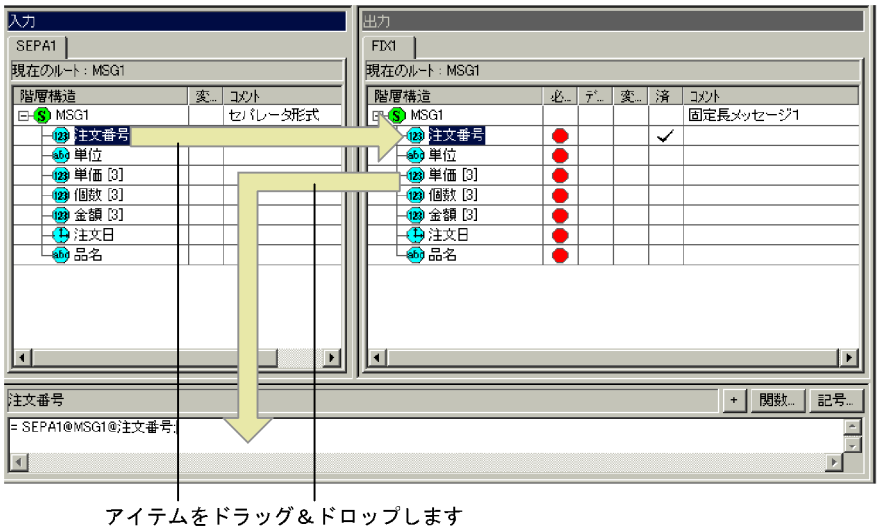
カラム名	意味
階層構造	ルート構造以下のコンポーネントをツリーで表示します。
必須	この欄にマークがあるコンポーネントに対して，必ずマップ式を定義する必要があることを表します。FDL エディタで，コンポーネントのプロパティにマップ式定義を必須とした場合はマークが表示され，必須としなかった場合マークは表示されません。 入力フォーマットツリーリストにこのカラムはありません。
デフォルト	この欄にマークがあるコンポーネントは，デフォルト式が指定されていることを表します。デフォルト式が指定されているコンポーネントに対してマップ式を定義すると，マップ式で定義した値が有効となります。 入力フォーマットツリーリストにこのカラムはありません。

カラム名	意味
変数式	この欄にマークがあるコンポーネントは，変数式が定義されていることを表します。
済	この欄にマークがあるコンポーネントに対してマップ式が定義済みであることを表します。マップ式が定義されているかどうかは，マップビューに表示されるテキスト文字の有無で判定されます。 入力フォーマットツリーリストにこのカラムはありません。
コメント	FDL エディタで型や構造に対して定義したコメントが表示されます。出力フォーマットツリーリストの場合，オプションの設定でコメントの代わりにマップ式を表示することもできます。

ツリーは次の動作によって縮小・展開できます。

- ツリーアイテムをダブルクリックします
- ツリーアイテムの左側に表示されている「＋」マーク又は「－」マークをクリックします

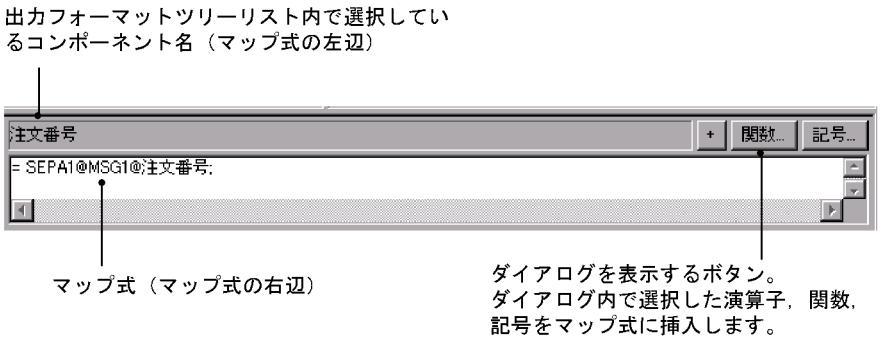
入力フォーマットツリーリストのアイテムを出力フォーマットツリーリストにドラッグ＆ドロップしたり，アイテムをマップビューにドラッグ＆ドロップしたりして，マッピングします。ただし，異なる MDL エディタ間でドラッグ & ドロップによるマッピングはできません。



(10) マップビュー

出力側のフォーマット内で選択しているコンポーネントのマップ式を表示するビューです。ここでは，式を直接記述したり，ダイアログから演算子や関数を選択したりして，マップ式を編集します。

図 4-4 マップビュー



また，フォーマットツリーリストのコンポーネントをマップビュー内にドラッグ＆ドロップすると，マップ式中にそのコンポーネント名を挿入できます。

(11) ステータスバー

メニューバーやツールバーからコマンドを選択する際に，コマンドの説明を表示するエリアです。

4.1.3 MDL エディタのコマンド一覧

MDL エディタのコマンドの一覧を表 4-2 に示します。

表 4-2 MDL エディタのコマンド一覧

メニュー名	メニューアイテム名	ショートカットキー	説明
ファイル (F)	新規作成 (N)	［Ctrl］＋［N］	MDL ドキュメントを新規に作成します。
	開く (O)...	［Ctrl］＋［O］	MDL ファイルを読み込みます。
	閉じる (C)	－	MDL ドキュメントウィンドウを閉じます。
	上書き保存 (S)	［Ctrl］＋［S］	MDL ドキュメントの内容を対応する MDL ファイルに上書き保存します。
	名前を付けて保存 (A)...	－	MDL ドキュメントに名前を付けて，MDL ファイルとして保存します。
	外部出力 (E)...	－	フォーマット情報又はマップ式情報をファイルに出力します。
	プロパティ (P)...	－	MDL ドキュメントのプロパティを表示します。
	（ファイル名）	－	最近使用したファイル（読み込んだファイル又は保存したファイル）を読み込みます。
	MDL エディタの終了 (X)	－	MDL エディタを終了します。

メニュー名	メニューアイテム名		ショートカットキー	説明
編集 (E)	削除 (D)	フォーマット (F)	－	入力又は出力フォーマットビュー内で選択したフォーマットを削除します。
		変数 (V)...	－	変数を削除します。
	マージ (M)...		－	FDL フォーマット，DTD フォーマット，又はマップ式をマージします。
	検索 (S)...		［Ctrl］＋［F］	データ名又は式文字列から，指定した文字列を検索します。
表示 (V)	ツールバー (T)		－	ツールバーの表示／非表示を切り替えます。
	ステータスバー (S)		－	ステータスバーの表示／非表示を切り替えます。
	下位コンポーネントだけを表示 (L)		－	指定したコンポーネントより下位のコンポーネントだけをフォーマットツリーリストに表示します。
	すべてのコンポーネントを表示 (A)		－	すべてのコンポーネントをフォーマットツリーリストに表示します。
挿入 (I)	フォーマット (F)...		－	入力又は出力フォーマットビューに，フォーマットを追加します。
	DTD フォーマット (D)...		－	入力又は出力フォーマットビューに，DTD フォーマットを追加します。
	変数 (V)...		－	入力又は出力フォーマットに，変数を追加します。
ツール (T)	マップ式 (E)...	演算子 (O)...	－	［演算子］ダイアログを表示します。
		関数 (F)...	－	［関数］ダイアログを表示します。
		特殊記号 (S)...	－	［特殊記号］ダイアログを表示します。
		値マッピング (V)...	－	［値マッピング］ダイアログを表示します。
	チェック (C)...		－	MDL ファイルの定義内容の正当性を検証します。
	マッピング確認 (M)...		－	入出力フォーマット間でのコンポーネントのマッピング情報を確認します。
	プロパティ (P)	フォーマット (F)...	－	選択しているフォーマットの詳細情報を表示します。
		入出力ファイル (I)...	－	選択しているフォーマットに対応する入出力ファイルを表示したり変更したりします。
		構造・型・属性 (S)...	－	選択している構造／型／ XML 属性の詳細情報を表示します。

メニュー名	メニューアイテム名		ショートカットキー	説明
		コンポーネント (C)...	－	選択しているコンポーネントの詳細情報を表示します。
		変数 (V)...	－	変数の詳細情報を表示します。
	プロパティ表示 (L)...		－	選択している要素のプロパティを一覧表示します。
	オプション (O)...		－	MDL エディタのオプションを設定します。
ウィンドウ (W)	下位コンポーネントを別ウィンドウに表示 (N)		－	選択したコンポーネントより下位のコンポーネントだけを別のドキュメントビューウィンドウのフォーマットツリーリストに表示します。
	重ねて表示 (C)		－	ドキュメントビューウィンドウのタイトルが重ならないように整列します。
	上下に並べて表示 (H)		－	ドキュメントビューウィンドウを上下に並べます。
	左右に並べて表示 (V)		－	ドキュメントビューウィンドウを左右に並べます。
	（ドキュメント名）		－	指定したドキュメントビューウィンドウをアクティブにします。
ヘルプ (H)	目次 (C)...		－	MDL エディタのヘルプの目次を表示します。
	検索 (S)...		－	MDL エディタのヘルプのトピック一覧を表示します。
	バージョン情報 (A)...		－	MDL エディタの形名，バージョンなどを表示します。

（凡例）

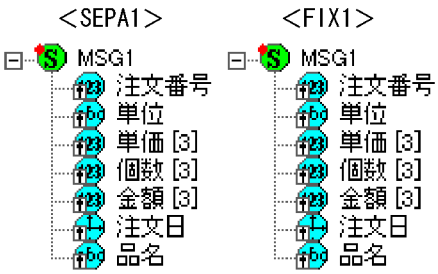
－：ショートカットキーはありません。

4.2 簡単なマッピング例

この節では，簡単なマッピングを定義しながら，MDL エディタの操作について説明します。

データの変換で一番多いのは，入力側の要素と出力側の要素とを 1 対 1 で対応付け，値を代入する方法です。FDL エディタの操作例で作成したフォーマットを使用して，値を代入するマッピングについて説明します。レングスタグ形式のフォーマット「SEPA1」のメッセージを，固定長形式のフォーマット「FIX1」のメッセージに変換するために，フォーマット間をマッピングし，MDL ファイルを作成します。データの構造を図 4-5 に示します。

図 4-5 フォーマットの構造



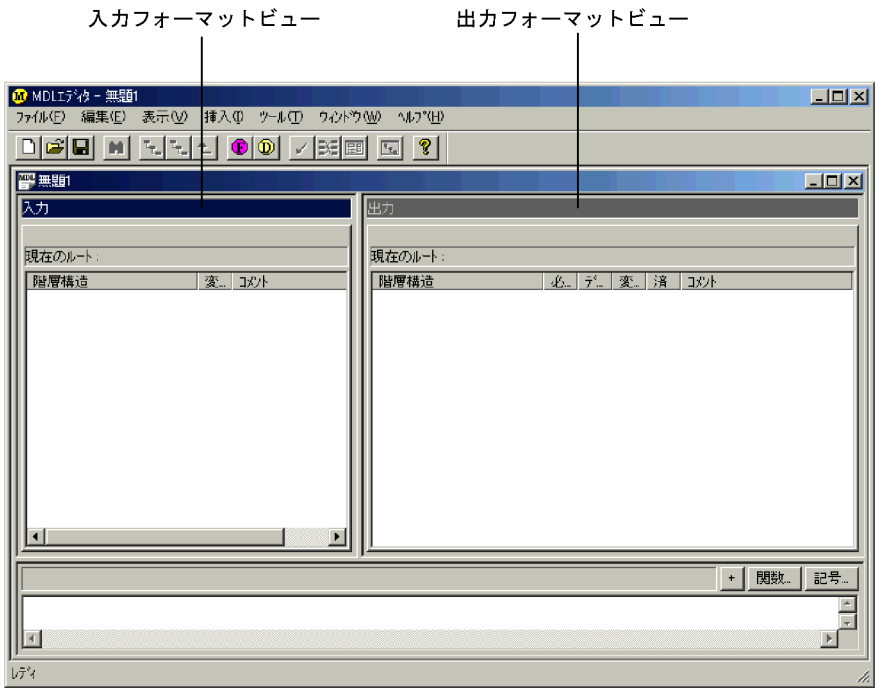
データの形式は異なりますが，要素名やツリー構造は同じです。

まず，フォーマットを読み込みます。

4.2.1 フォーマットの挿入

マッピングするフォーマットを読み込む MDL ドキュメントウィンドウを表示します。

1. **［ファイル］－［新規作成］を選択します**
   MDL ドキュメントウィンドウが表示されます。ドキュメントウィンドウ内に，マッピングする入力側，出力側のフォーマットを読み込みます。

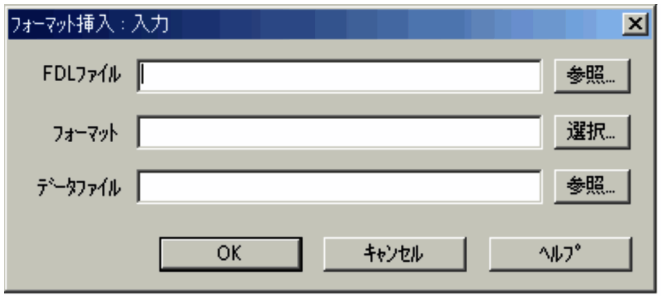


まず，入力側のフォーマットを読み込みます。

2. **入力フォーマットビューを選択します**

3. **［挿入］－［フォーマット］を選択します**
［フォーマット挿入：入力］ダイアログが表示されます。なお，Windows エクスプローラを開き，挿入したい FDL ファイルをフォーマットビューにドラッグ & ドロップして挿入することもできます。

図 4-6 ［フォーマット挿入：入力］ダイアログ



読み込むフォーマットを定義している FDL ファイル名，読み込むフォーマット名，入力データのファイル名を指定するダイアログです。

4. **入力側のフォーマットの FDL ファイル名を指定します**

FDL ファイル名は直接入力できますが，［参照］ボタンをクリックすると，FDL ファイルを参照して指定できます。選択できるフォーマットが一つだけの場合は，フォーマット欄にフォーマット名が自動的に入力されます。その他の場合は，［フォーマット選択］ダイアログが表示されます。

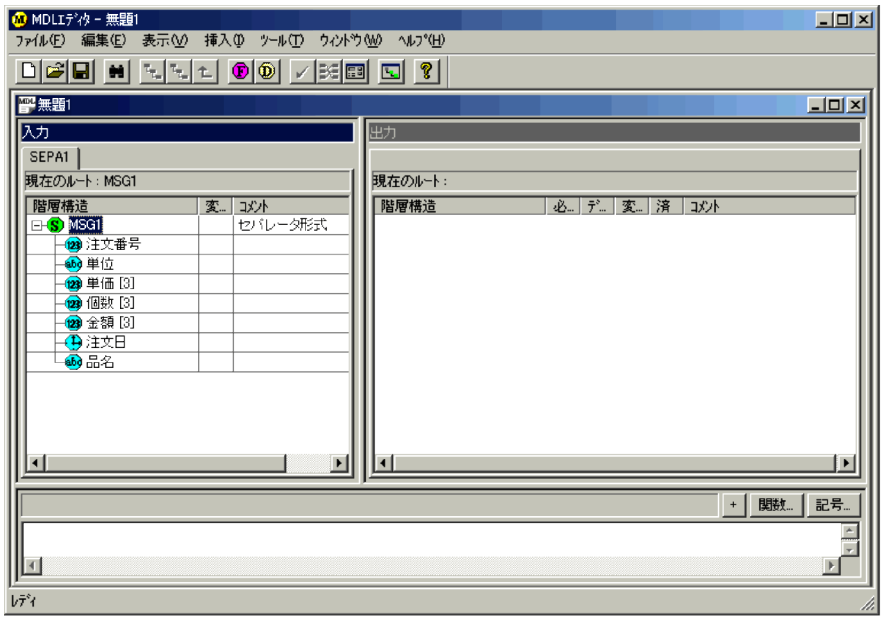
図 4-7 ［フォーマット選択］ダイアログ

このダイアログには，指定した FDL ファイルに定義されているフォーマットが表示されます。ここでは，入力フォーマットビューにフォーマットを挿入するので，入力側のフォーマットだけが表示されます。

5. **フォーマットを選択して［OK］ボタンをクリックします**
   ここでは，入力フォーマットとして「SEPA1」を指定します。
   選択したフォーマット名が［フォーマット挿入：入力］ダイアログのフォーマット名に挿入されます。

6. **入力データファイル名を指定します**
   指定したフォーマットに対応する入力データのファイル名を指定します。入力データファイル名は直接入力できますが，［参照］ボタンをクリックすると既存のファイル名を参照して指定できます。入力データファイル名は，データを変換するときのコマンドの引数で指定できるので，このダイアログでは未指定でもかまいません。

7. **［OK］ボタンをクリックします**
   入力フォーマットビューに，選択したフォーマットが挿入されます。



次に，出力フォーマットを読み込みます。

8. **出力フォーマットビューを選択します**

9. **［挿入］－［フォーマット］を選択します**

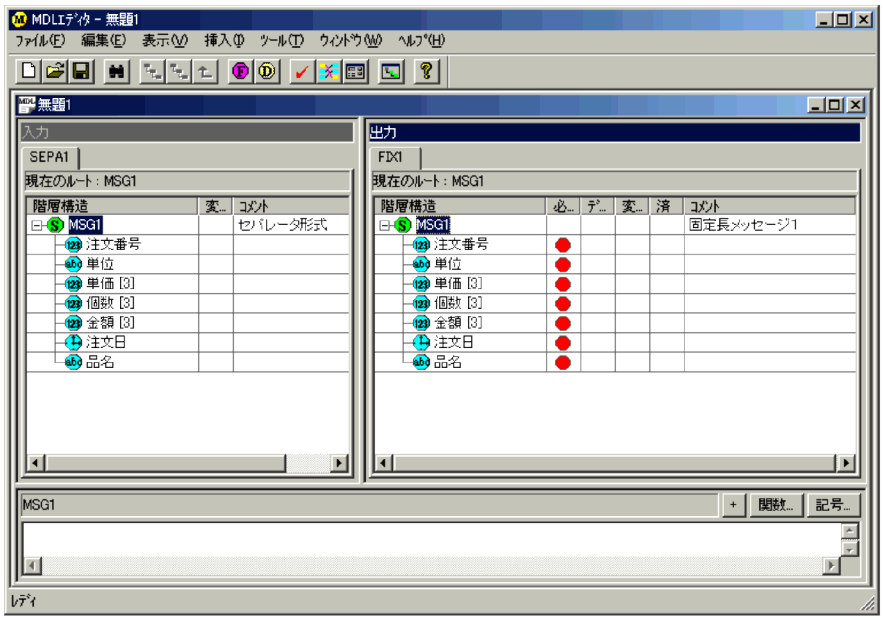
   なお，Windows エクスプローラを開き，挿入したい FDL ファイルをフォーマットビューにドラッグ & ドロップして挿入することもできます。

   どちらの場合も［フォーマット挿入：出力］ダイアログが表示されます。このダイアログに，出力側のフォーマットの FDL ファイル名，フォーマット名，出力データファイル名を設定します。指定方法は入力側のフォーマットの指定と同様です。出力データファイル名も，データを変換するときのコマンドの引数で指定できるので，このダイアログでは未指定でもかまいません。なお，出力データに入力データと同じファイル名を指定するとエラーになるので，入力データと異なるファイル名を指定してください。

   ここでは，出力フォーマットとして「FIX1」を指定します。

10. **［OK］ボタンをクリックします**

    出力フォーマットビューに，選択したフォーマットが挿入されます。

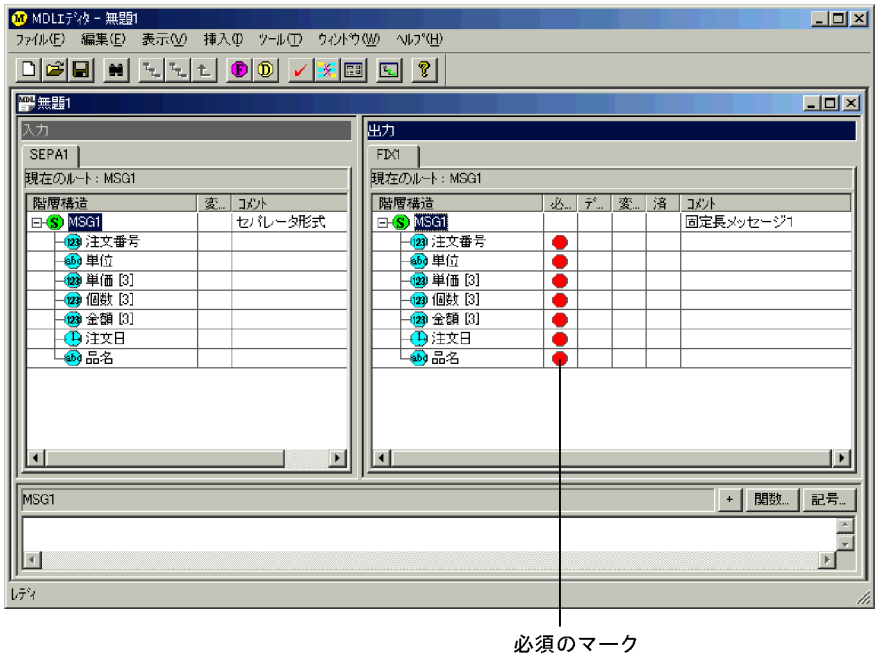


入力，出力それぞれのフォーマットビューには，ルート構造以下のコンポーネントがツリーで表示されます。フォーマットの各コンポーネントの内容は，プロパティダイアログで確認できます。以上でマッピングのための環境が整いました。

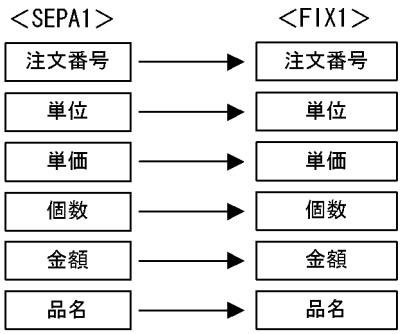
次に，入力側と出力側のコンポーネントを対応付け，出力側のコンポーネントに対してマップ式を定義します。

4.2.2 マップ式の定義

マップ式を定義する必要があるのは，出力フォーマットビューで，マップ式の定義が必須であることを示すマークが表示されているコンポーネント，可変回出現コンポーネント，及び省略できない型コンポーネントです。



ここでは，出力側のルート構造「MSG1」の子コポーネントすべてにマップ式を定義する必要があります。フォーマット「SEPA1」とフォーマット「FIX1」のコンポーネント間の対応を次に示します。



フォーマット「FIX1」の「注文日」以外の要素は，入力側のフォーマット「SEPA1」のコンポーネントを，出力側のフォーマット「FIX1」の同じ名前のコンポーネントに代入するマップ式を記述します。「注文日」は，入力側の同じ名前のコンポーネントを代入するのではなく，データ変換時の日付が出力されるように設定したいので，関数を使用したマップ式を記述します。

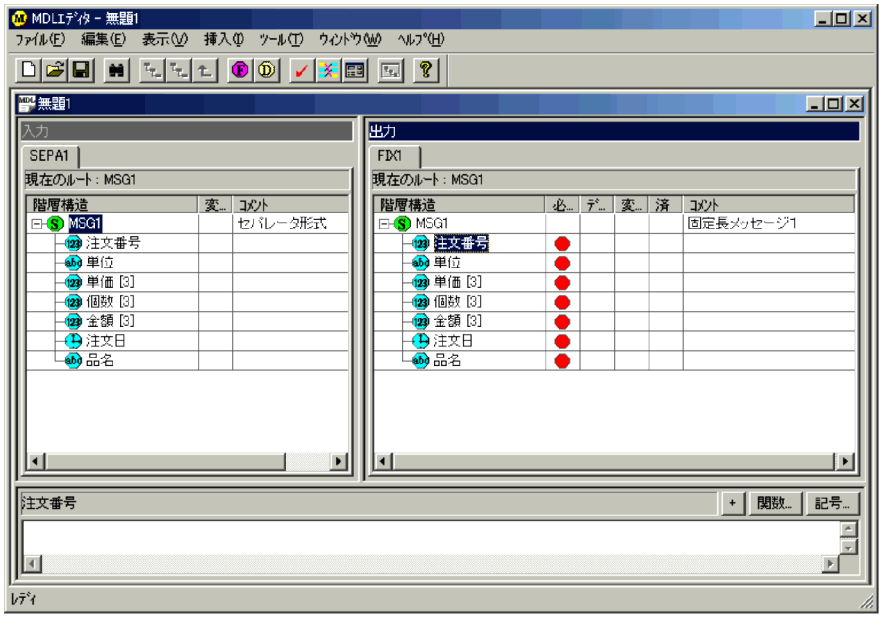
要素ごとにマップ式の記述について説明します。

(1) 要素「注文番号」「単位」「品名」のマップ式

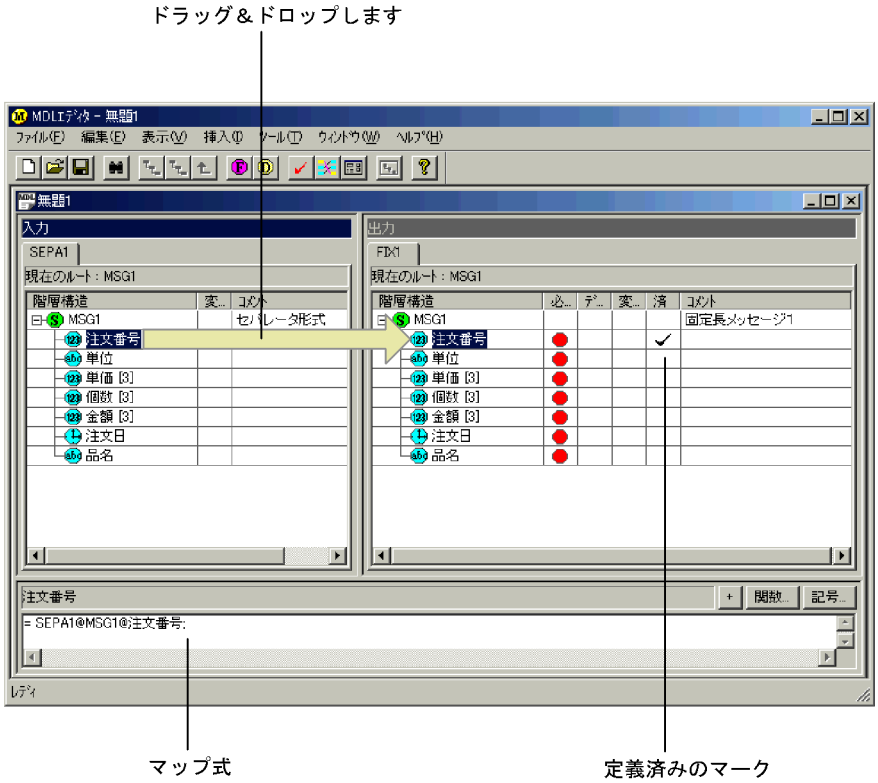
出力フォーマットのコンポーネントに，対応する入力フォーマットのコンポーネントをドラッグ＆ドロップし，代入のマップ式を定義します。

まず，要素「注文番号」に代入のマップ式を記述します。

1. **出力フォーマットの「注文番号」をクリックして選択します**



2. **入力フォーマットの「注文番号」をクリックして選択し，「注文番号」を出力フォーマットの「注文番号」へドラッグ＆ドロップします**



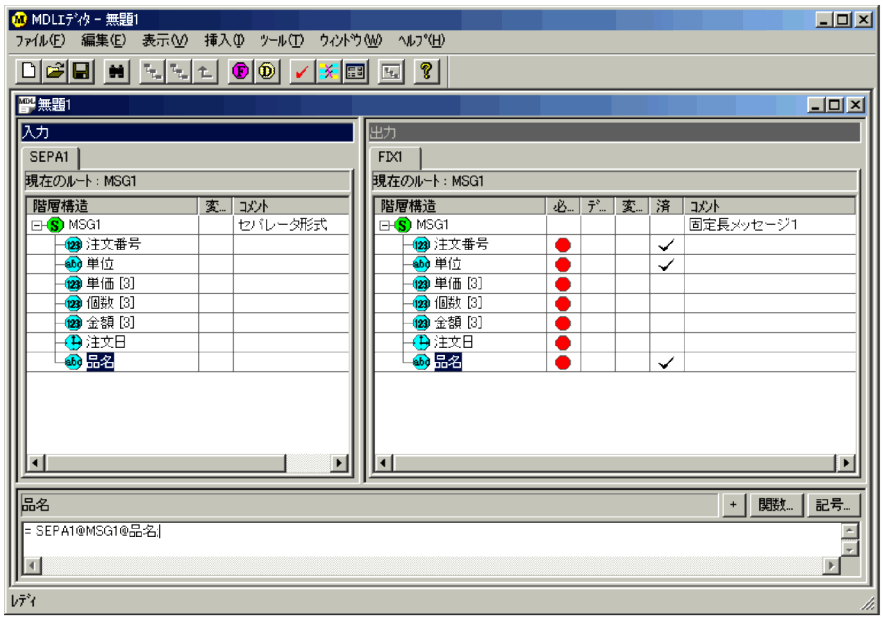
出力フォーマットの「注文番号」にはマップ式が定義済みであることを示すマークが付き，マップビューには出力フォーマットの「注文番号」のマップ式が表示されます。マップ式を次に示します。

```
= SEPA1@MSG1@注文番号 ;
```

代入のマップ式には，ドラッグ＆ドロップしたコンポーネント名が表示されます。コンポーネント名は，フォーマット名，ルート構造から目的のコンポーネントまでを「@」でつないだグローバル名で表します。ここでは，出力フォーマットの「注文番号」には，入力フォーマット「SEPA1」のルート構造「MSG1」の子コンポーネント「注文番号」を代入する，という意味のマップ式です。

3. **「単位」及び「品名」のマップ式を記述します**

「注文番号」と同様に，対応する入力フォーマットのコンポーネントをドラッグ＆ドロップし，代入のマップ式を記述します。

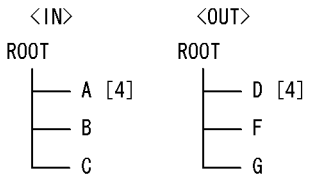


(2) 要素「単価」「個数」「金額」のマップ式

要素「単価」「個数」「金額」は複数回出現するコンポーネントです。入力側と出力側のコンポーネントを対応付けて代入のマップ式を記述するときに，出現順序を対応付ける INDEX 関数を使用します。

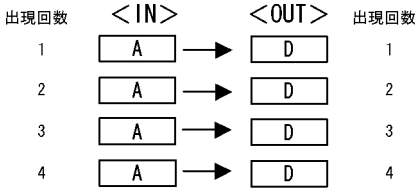
● INDEX 関数を使用したマップ式

INDEX 関数は，引数で指定したコンポーネントの現在の出現番号（インデクスと呼びます）を取り出し，コンポーネント間の出現順序を 1 対 1 で対応付けます。例えば，互いに 4 回出現する入力側のコンポーネント「A」と出力側のコンポーネント「D」を対応付けたいとします。



（凡例）　[ ]内は出現回数です。

1 回目に出現した「A」の値は 1 回目に出現した「D」の値として代入し，2 回目に出現した「A」の値は 2 回目に出現した「D」の値に代入する，というようにコンポーネント間の値の代入を出現回数で対応付けます。

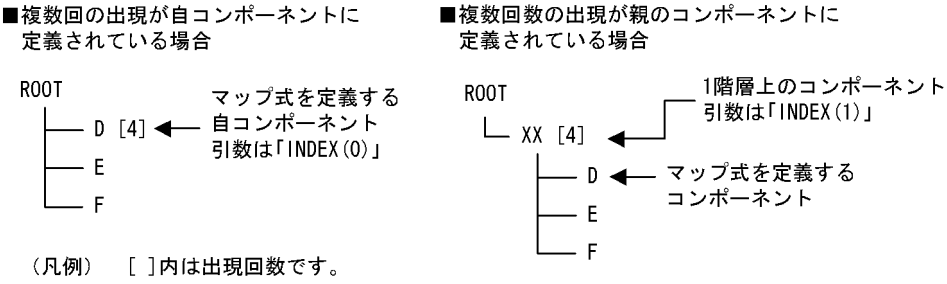


この場合，コンポーネント「D」に対して INDEX 関数を使用したマップ式を記述します。入力側のコンポーネント「A」を出力側のコンポーネント「D」にドラッグ＆ドロップで代入すると，次のようなマップ式が表示されます。

```
=IN@ROOT@A[INDEX(  )] ;
```

INDEX 関数の括弧内の引数は，式が記述されているコンポーネントの位置を起点として階層数を表すもので，インデクス値の参照先となるコンポーネントを指定します。「0」はマップ式を記述する自コンポーネント，「n」はマップ式を記述するコンポーネントから n 階層分，上位に位置するコンポーネントを表します。INDEX 関数の引数の指定例を図 4-8 に示します。

図 4-8　INDEX 関数の引数の指定例



コンポーネント「D」は自コンポーネントに複数回の出現回数が定義されているので，引数で「0」を指定して自コンポーネントのインデクスを取り出し，その同じインデクスのコンポーネント「A」の値を代入します。マップ式を次に示します。

```
=IN@ROOT@A[INDEX(0)] ;
```

INDEX 関数で，自コンポーネントである「D」のインデクスを取り出します。インデクスが「1」の場合は，1 回目に出現したコンポーネント「A」の値が 1 回目に出現したコンポーネント「D」に代入されます。同様に，インデクスに対応した値が代入されます。

「INDEX(0)」と記述する代わりに，入力側と出力側の出現順序を対応付ける略記法「?」も使用できます。その場合のマップ式を次に示します。

```
=IN@ROOT@A[?] ;
```

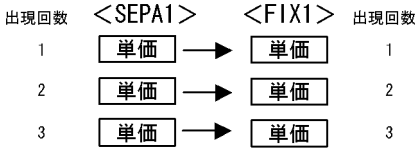
では，実際に INDEX 関数を使用したマップ式を記述します。

まず，要素「単価」のマップ式を記述します。

1. **出力フォーマットの「単価」を選択し，入力フォーマットの「単価」をドラッグ＆ドロップします**
   マップビューに次のマップ式が表示されます。

   ```
   = SEPA1@MSG1@単価 [INDEX(  )] ;
   ```

   「単価」は，3 回出現するコンポーネントなので，マップ式には INDEX 関数を使用します。入力側のコンポーネント「単価」と出力側のコンポーネント「単価」を 1 対 1 で出現順序を対応付けます。



   ここでは，自コンポーネントである出力側のコンポーネント「単価」を指定したいので，INDEX 関数の引数に「0」を記述します。出力側のコンポーネント「単価」のインデクスを取り出し，入力側と出力側の「単価」の出現順序を対応付けるように指定します。

2. **カーソルが INDEX 関数の括弧内に位置づけられているので，括弧内に「0」と入力します**
   マップ式を次に示します。

   ```
   = SEPA1@MSG1@単価 [INDEX(0)] ;
   ```

   これで，入力側と出力側の「単価」の出現順序が対応付けられました。

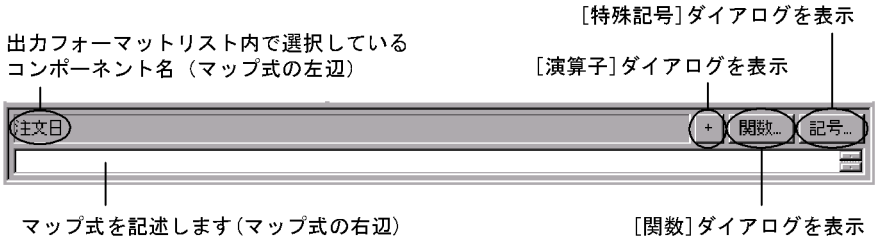
3. **「個数」及び「金額」のマップ式を記述します**
   「単価」と同様に，INDEX 関数を使用したマップ式を記述します。

(3) 要素「注文日」のマップ式

要素「注文日」は，入力フォーマットの「注文日」を代入するのではなく，データを変換したときの日付が出力されるように，関数を使用してマップ式を記述します。

1. **出力フォーマットの「注文日」を選択します**
   ここでは，コンポーネントをドラッグ＆ドロップするのではなく，マップビューに式を直接記述します。マップ式で使用する演算子，関数，及び特殊記号は直接入力できますが，ダイアログからも選択できます。ダイアログから選択した演算子や関数は，

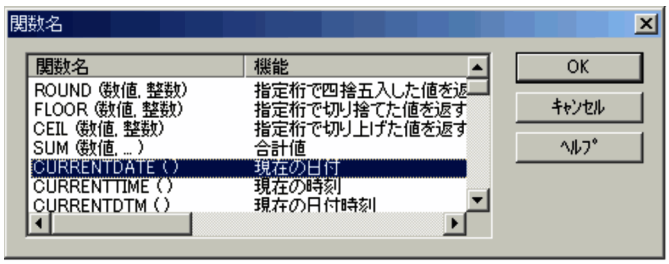
マップビュー内のカーソル位置に挿入されます。



マップ式の左辺には，マップ式を定義するコンポーネント名が表示されます。ここでは，日付を設定するために関数を使用します。

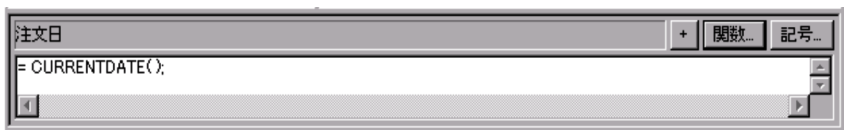
2. **［関数］ボタンをクリックします**
［関数名］ダイアログが表示されます。

図 4-9 ［関数名］ダイアログ



3. **一覧から関数を選択します**
ここでは，データを変換するときの日付を「注文日」に出力したいので，現在の日付を返す関数「CURRENTDATE」を選択します。

4. **［OK］ボタンをクリックします**
マップビューに選択した関数が挿入されます。



要素「注文日」のマップ式を次に示します。

```
= CURRENTDATE();
```

CURRENTDATE 関数には引数は必要ないので，ここではこれ以上記述する必要はありません。要素「注文日」には，データを変換するときの日付が出力されるようになりました。

以上ですべての要素にマップ式が定義できました。

マップ式の記述法として，マップ式左辺の先頭に代入式のイコール「=」，マップ式の最後にセミコロン「;」が必要です。マップ式を直接入力して記述する場合は，「=」，「;」の記述もれがないように注意してください。

各コンポーネントのマップ式を表 4-3 に示します。

表 4-3　各コンポーネントのマップ式（簡単なマッピング例）

要素	マップ式
注文番号	= SEPA1@MSG1@ 注文番号 ;
単位	= SEPA1@MSG1@ 単位 ;
単価 [3]	= SEPA1@MSG1@ 単価 [INDEX(0)] ;
個数 [3]	= SEPA1@MSG1@ 個数 [INDEX(0)] ;
金額 [3]	= SEPA1@MSG1@ 金額 [INDEX(0)] ;
注文日	= CURRENTDATE() ;
品名	= SEPA1@MSG1@ 品名 ;

4.2.3　MDL の検証

データを変換する前に，チェックコマンドを実行し，マップ式などの定義内容にエラーがないことを確認する必要があります。

1. **［ツール］－［チェック］を選択します**
   マップ式などの定義内容の正当性を検証します。検証結果は，［結果］ダイアログに一覧表示されます。

図 4-10　［結果］ダイアログ

エラーがある場合は，一覧でエラー内容をダブルクリックすると，エラーのあるデータがツリービュー内で選択状態になります。エラーを修正したら，再度検証を実行し，エラーがないことを確認してください。

2. **エラーがなければ，［閉じる］ボタンをクリックして［結果］ダイアログを閉じます**

4.2.4 MDL ファイルの保存

マップ式の定義内容にエラーがないことを確認したら，MDL に名前を付けて保存します。

1. **［ファイル］－［名前を付けて保存］を選択します**
   ［名前を付けて保存］ダイアログが表示されるので，ファイル名を指定します。
2. **［OK］ボタンをクリックします**
   以上で入出力フォーマット間をマッピングした MDL ファイルが作成できました。

4.3 複雑なマッピング例

この節では，複雑なマッピングを定義するときの，マップ式の定義について説明します。説明する内容を次に示します。

- 構造を含んだマッピング
- レングスタグフォーマットへのマッピング
- 値を持つ要素のマッピング
- 条件式と NULL を使用したマッピング
- 選択構造を持つフォーマットのマッピング

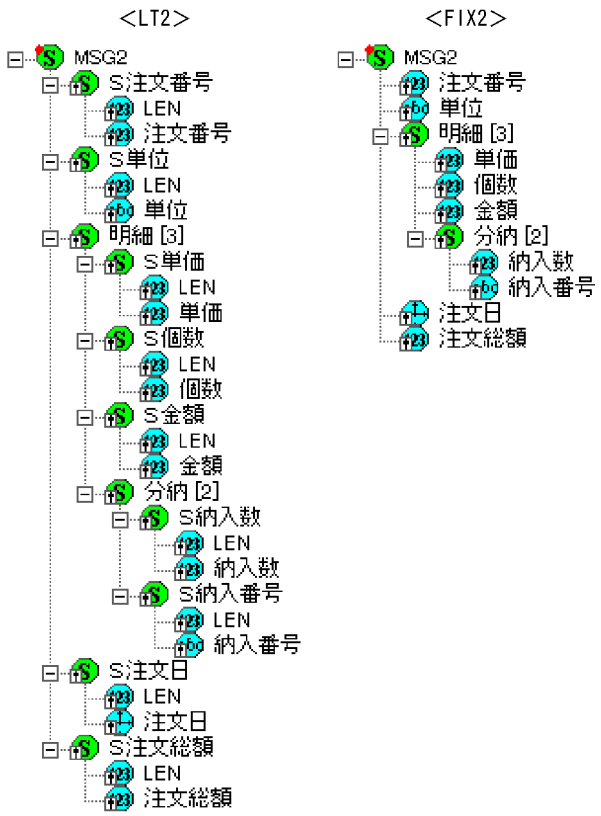
各内容について次に説明します。

4.3.1 構造を含んだマッピング

ここでは，レングスタグ形式のフォーマットから固定長形式のフォーマットへマッピングする例を使用し，構造が階層化されたデータのマッピングについて説明します。

レングスタグ形式のフォーマットとして「LT2」，固定長形式のフォーマットとして「FIX2」を使用します。データの構造を図 4-11 に示します。

図 4-11　フォーマット「LT2」「FIX2」のデータの構造

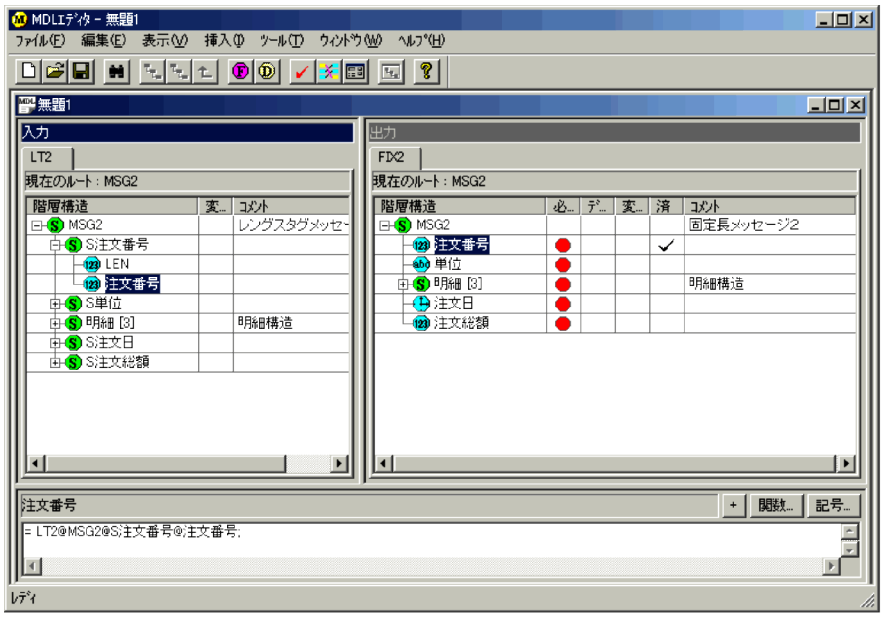


入力側のデータのレングスタグ構造には，各要素のサイズを表す要素「LEN」がありますが，出力側のデータにはこの要素は必要ありません。したがって，マップ式は，レングスタグ構造の実際の値を持つ要素だけを取り出すマップ式を記述します。

要素ごとにマップ式の定義について説明します。

(1) 要素「注文番号」「単位」のマップ式

入力フォーマットの「注文番号」を出力フォーマットの「注文番号」にドラッグ＆ドロップし，代入のマップ式を記述します。



マップ式を次に示します。

```
= LT2@MSG2@S注文番号@注文番号 ;
```

要素「単位」も要素「注文番号」と同様に代入のマップ式を記述します。

(2) 構造「明細」のマップ式

マップ式は，構造に対しても記述できます。構造に対するマップ式は，入力側と出力側それぞれの出現順序の関係を対応付けます。要素のマップ式と同様に，入力側の構造を出力側の構造へドラッグ＆ドロップして記述できます。複数回出現する構造には，複数回出現する要素のマップ式の記述と同様に，INDEX 関数を使用したマップ式を記述します。

構造「明細」のマップ式を記述します。出力側の構造「明細」は，入力側の構造「明細」とマッピングします。

1. **出力側の構造「明細」を選択し，入力側の構造「明細」をドラッグ＆ドロップします**
   マップ式を次に示します。

   ```
   = LT2@MSG2@明細[INDEX(  )] ;
   ```

   構造「明細」は，複数回出現するので，INDEX 関数を使用します。出力側の構造「明細」自身を指定したいので，引数には自コンポーネントを表す「0」を指定します。
2. **INDEX 関数の括弧内に引数として「0」を記述します**

これで入力側の構造「明細」と出力側の構造「明細」が対応付けられました。

(3) 構造「明細」の子コンポーネントのマップ式

構造「明細」の子コンポーネントのマップ式を記述します。

親の構造「明細」のマップ式で入出力の関係が定義されているので，子コンポーネントの式には構造「明細」のマップ式を引き継ぐ形のマップ式を記述します。

まず，要素「単価」は，入力側の構造「明細」の子コンポーネント「単価」を代入するので「入力側の単価要素を代入する」という意味の式を記述します。

1. **出力フォーマットの「単価」を選択します**
   入力側の「単価」を選択して，出力側の「単価」へドラッグ＆ドロップします。
   この場合のマップ式を次に示します。

   ```
   = $@S単価@単価 ;
   ```

   「$」は親コンポーネントの構造で定義してあるマップ式を引き継ぐための記述法です。ここでは，親コンポーネントの構造「明細」は，入力側の構造「明細」と対応付けられています。したがって，マップ式は，入力側の構造「明細」の孫コンポーネント「単価」の値を代入する，という意味になります。既に親コンポーネントの構造同士でインデクスが対応付けられているので，ここではインデクスについて記述する必要はありません。
   なお，ドラッグ＆ドロップで親コンポーネントのマップ式を引き継ぐ場合，親（先祖）のコンポーネントのマップ式が正しく記述されている必要があります。親（先祖）のコンポーネントのマップ式が記述されていない，又は誤りがある場合は，$ を使用して引き継ぐ形のマップ式は記述しません。
2. **「単価」と同様に，要素「個数」「金額」，構造「分納」も親コンポーネントのマップ式を引き継ぐマップ式を記述します**
   構造「分納」は複数回出現するので，INDEX 関数を使用してインデクスを対応付けます。

   ```
   = $@分納[INDEX(0)] ;
   ```

   引数には自コンポーネントを表す「0」を指定します。

(4) 構造「分納」の子コンポーネントのマップ式

分納構造の子コンポーネント「納入数」「納入番号」は「$」を使用して，親コンポーネントの構造「分納」のマップ式を引き継ぐ形式のマップ式を記述します。

```
= $@S納入数@納入数;
```

(5) 要素「注文日」のマップ式

要素「注文日」は，「4.2　簡単なマッピング例」での例と同様に，データ変換時の日付を出力するようにしたいので，CURRENTDATE 関数のマップ式を記述します。

```
= CURRENTDATE();
```

(6) 要素「注文総額」のマップ式

要素「注文総額」は，出力した構造「明細」の「金額」の合計を出力する要素です。したがって，出力した構造「明細」の「金額」の合計を計算するマップ式を記述します。ここでは関数を使用します。

1. **出力側の「注文金額」を選択します**
2. **［関数］ボタンをクリックして［関数］ダイアログを表示します**
3. **一覧から，要素の合計値を計算する SUM 関数を選択します**
   マップ式を次に示します。

   ```
   = SUM(  ) ;
   ```

4. **SUM 関数の引数として，合計値を計算する要素のグローバル名を記述します**
   ここでは，出力側の「金額」の合計を計算するので，引数に「FIX2@MSG2@ 明細 [*]@ 金額」と記述します。インデクス「*」は，すべての構造の出現を意味する記述法です。ここでは，すべての構造「明細」の出現を意味します。つまり，明細 [1]，明細 [2] 及び明細 [3] の「金額」をすべて合計する，という意味になります。なお，インデクス「*」は，SUM 関数などの特定の関数の引数だけで使用できる略記法です。
   「注文総額」のマップ式を次に示します。

   ```
   = SUM( FIX2@MSG2@明細[*]@金額 ) ;
   ```

   また，要素「金額」は 3 回しか現れないので，四則演算でマップ式を記述してもかまいません。四則演算を使用した要素「金額」マップ式を次に示します。

   ```
   = FIX2@MSG2@明細[1]@金額 + FIX2@MSG2@明細[2]@金額 + FIX2@MSG2@明細
   [3]@金額 ;
   ```

   SUM 関数のマップ式と同じ結果が得られます。
   以上で，レングスタグ形式のフォーマット「LT2」と，固定長形式のフォーマット「FIX2」のマッピングが定義できました。
   各コンポーネントのマップ式を表 4-4 に示します。

表 4-4 各コンポーネントのマップ式（構造を含んだマップ例）

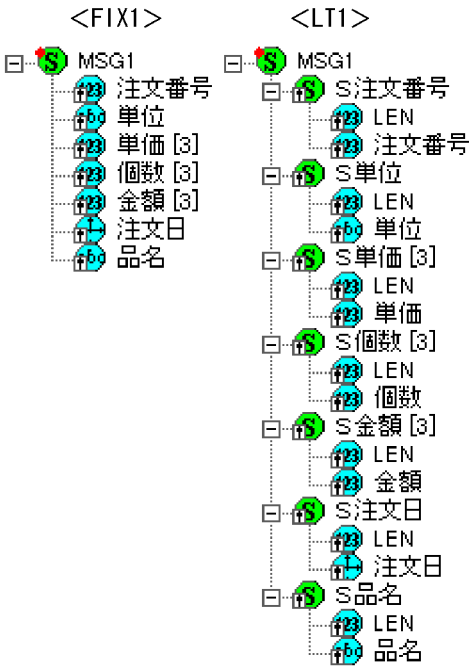
構造／要素			マップ式
注文番号			= LT2@MSG2@S 注文番号 @ 注文番号 ;
単位			= LT2@MSG2@S 単位 @ 単位 ;
明細			= LT2@MSG2@ 明細 [INDEX(0)] ;
	単価		= $@S 単価 @ 単価 ;
	個数		= $@S 個数 @ 個数 ;
	金額		= $@S 金額 @ 金額 ;
	分納		= $@ 分納 [INDEX(0)] ;
		納入数	= $@S 納入数 @ 納入数 ;
		納入番号	= $@S 納入番号 @ 納入番号 ;
注文日			= CURRENTDATE() ;
注文総額			= SUM(FIX2@MSG2@ 明細 [*]@ 金額) ;

4.3.2 レングスタグフォーマットへのマッピング

ここでは，固定長形式のフォーマットからレングスタグ形式のフォーマットへのマッピングについて説明します。

固定長形式のフォーマットとして「FIX1」，レングスタグ形式のフォーマットとして「LT1」を使用します。データの構造を図 4-12 に示します。

図 4-12 フォーマット「FIX1」「LT1」のデータの構造



レングスタグ形式の構造は，要素サイズを表す要素（ここでは要素「LEN」）があります。要素「LEN」には，直接対応する要素が入力側にありません。入力データの形式が固定長の場合は，入力側の要素のサイズは決まっているので，レングスタグのサイズを表す要素に対して，固定値を代入することでマップ式を定義できます。

しかし，注意点として，入力側の要素のデータから不要なスペースなどを削除してデータを出力する場合は，データの内容によって出力される要素のサイズが異なってしまいます。また，入力側の要素が可変長の場合も同様の問題が発生します。

この対策として，レングスタグ構造の要素サイズを表す要素には，実際に出力要素として代入された要素のサイズを代入するマップ式を記述する必要があります。ここでは，マップ式の評価を遅延（待機）させる指定「%delay」とコンポーネントのサイズを計算する関数「LENGTH」を使用します。

(1) 評価順序を遅延（待機）させるマップ式

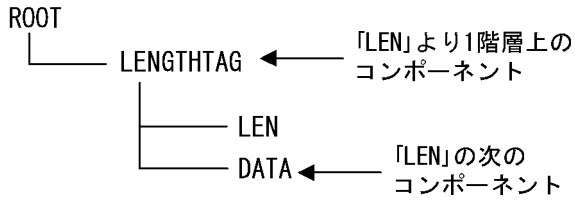
普通，マップ式は出力側の要素の出現順に評価されます。また，前提として，マップ式が評価される時点では，式中に現れるすべてのコンポーネントの値が決まっていなければいけません。この場合，式に記述するコンポーネントが入力側の要素であれば問題ありませんが，レングスタグ構造では出力側の要素を指定する必要があります。

例えば，要素「LEN」に対する式を評価する時点では，まだ要素「DATA」の値は決まっていないため，単純にマップ式中に要素「DATA」を記述すると，式の評価結果はエラーになってしまいます。

要素「DATA」の値が決まるまで，要素「LEN」のマップ式評価を待機させる必要があります。このために，評価順序を遅延（待機）させる「%delay」を利用します。評価順序遅延指定には次の 2 種類があります。

- 自コンポーネントの次のコンポーネントが評価されるまでマップ式の評価を待機させる指定（%delay %next)。
- n 階層上の親の子コンポーネントがすべて評価されるまでマップ式の評価を待機させる指定（%delay n)。%delay を指定するコンポーネントの位置を起点として，階層数を指定します。INDEX 関数のときと同様「0」は自コンポーネントを表し，「n」は n 階層分，上位に位置するコンポーネントを表します。

フォーマット「OUT」



要素「LEN」の次のコンポーネントである「DATA」が評価されるまで，要素「LEN」のマップ式評価を遅延（待機）させるには，「%delay %next」を指定します。同時に，LENGTH 関数を使用してサイズを計算します。LENGTH 関数は，引数で指定されたコンポーネントのサイズを計算する関数です。レングスタグ構造の下の要素で，実際の値を持つ要素を指定します。ここでは，実際の値を持つ要素として「DATA」を指定します。

要素「LEN」に対するマップ式を次に示します。

```
%delay %next =LENGTH(OUT@ROOT@LENGTHTAG@DATA);
```

要素「DATA」のサイズが計算されるまで，要素「LEN」のマップ式定義を遅延させる，という指定になります。

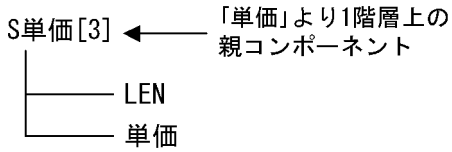
評価順序遅延指定は，同じ親コンポーネントを持つ兄弟コンポーネントの中で複数指定できません。また，評価順序遅延指定に誤りがあると，重大なエラーの原因となるおそれがあるので注意してください。

(2) 代入のマップ式

レングスタグ構造の実際のデータを持つ要素は，対応する入力側のコンポーネントをドラッグ＆ドロップし，代入のマップ式を記述します。構造「S 注文番号」の子コンポーネント「注文番号」のマップ式を次に示します。

```
= FIX1@MSG1@注文番号;
```

また，複数回出現する構造の子コンポーネントの場合は，INDEX 関数を使用した代入のマップ式を記述します。構造「S 単価」の子コンポーネント「単価」は 3 回出現するコンポーネントですが，複数回出現は親コンポーネントに定義されています。



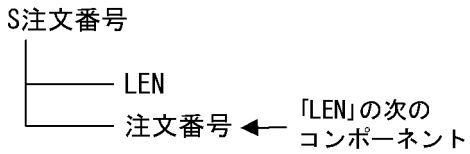
したがって，コンポーネント「S 単価」のインデクスを取り出して，出現順序を対応付ける必要があります。INDEX 関数の引数に，出力側のコンポーネント「単価」の 1 階層上のコンポーネントを表す「1」を記述し「S 単価」を指定します。この場合のマップ式を次に示します。

```
= FIX1@MSG1@単価[INDEX(1)] ;
```

(3) %delay のマップ式

次に，要素「LEN」マップ式を記述します。

ここでは，構造「S 注文番号」の子コンポーネント「LEN」のマップ式を記述します。「LEN」は次のコンポーネント「注文番号」の値を代入するので，「注文番号」の値が決まるまでマップ式評価を遅延させる必要があります。



そこで，評価順序を遅延させる「%delay」を指定します。

1. **構造「S 注文番号」の子コンポーネント「LEN」を選択します**
   マップビューに式を記述します。

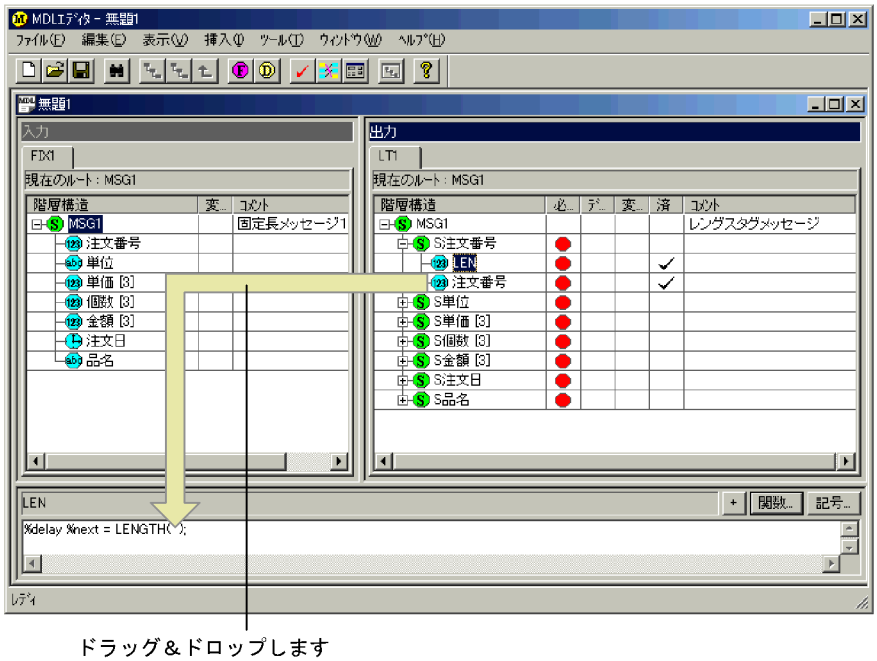
2. **マップ式に評価遅延指定「%delay %next」を記述します**
   「%delay %next」はテキストに直接記述するか又は［特殊記号］ダイアログで選択してマップ式に挿入します。

3. **マップ式に LENGTH 関数を記述します**
   テキストに直接記述するか又は［関数］ダイアログで選択してマップ式に挿入します。マップ式を次に示します。

```
%delay %next = LENGTH(  ) ;
```

4. **LENGTH 関数の引数として，括弧内に実際の値を持つ要素を指定します**
   構造「S 注文番号」の子コンポーネント「注文番号」を指定します。

5. **出力側の「注文番号」をマップ式の括弧内にドラッグ＆ドロップします**



すると，次のマップ式が表示されます。

```
%delay %next = LENGTH(LT1@MSG1@S注文番号@注文番号 ) ;
```

また，複数出現する構造の子コンポーネントのマップ式定義では，INDEX 関数を使用します。構造「S 単価」の子コンポーネント「LEN」のマップ式を次に示します。

```
%delay %next = LENGTH( LT1@MSG1@S単価[INDEX(1)]@単価 ) ;
```

INDEX 関数の引数には，構造「S 単価」を指定したいので，1 階層上のコンポーネントを表す「1」を指定します。

また，略記法を用いると，構造「S 単価」の子コンポーネント「LEN」のマップ式は次のようになります。

```
%delay %next = LENGTH( ¥1@単価 ) ;
```

「¥n」は n 階層上の親の構造を表すので，この場合「¥1」は「注文番号」の親構造「S 注文番号」になります。

以上で，固定長形式のフォーマット「FIX1」とレングスタグ形式のフォーマット「LT1」のマップ式が定義できました。

各コンポーネントのマップ式を表 4-5 に示します。

表 4-5　各コンポーネントのマップ式（レングスタグフォーマットへのマッピング例）

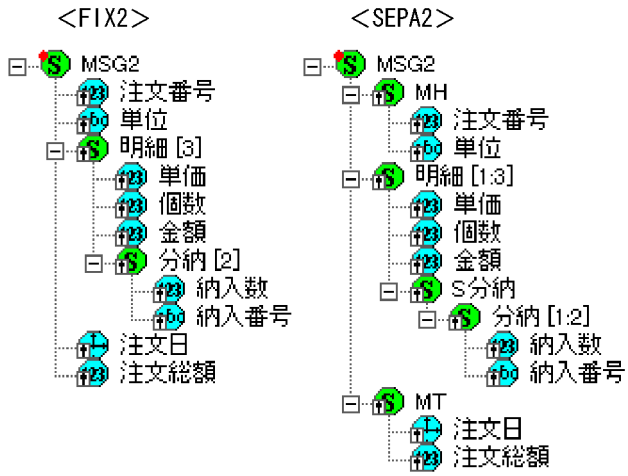
構造／要素		マップ式	
S 注文番号	LEN	%delay %next	= LENGTH(¥1@ 注文番号);
	注文番号		= FIX1@MSG1@ 注文番号 ;
S 単位	LEN	%delay %next	= LENGTH(¥1@ 単位);
	単位		= FIX1@MSG1@ 単位 ;
S 単価 [3]	LEN	%delay %next	= LENGTH(¥1@ 単価);
	単価		= FIX1@MSG1@ 単価 [INDEX(1)];
S 個数 [3]	LEN	%delay %next	= LENGTH(¥1@ 個数);
	個数		= FIX1@MSG1@ 個数 [INDEX(1)];
S 金額 [3]	LEN	%delay %next	= LENGTH(¥1@ 金額);
	金額		= FIX1@MSG1@ 金額 [INDEX(1)];
S 注文日	LEN	%delay %next	= LENGTH(¥1@ 注文日);
	注文日		= FIX1@MSG1@ 注文日 ;
S 品名	LEN	%delay %next	= LENGTH(¥1@ 品名);
	品名		= FIX1@MSG1@ 品名 ;

4.3.3　値を持つ要素のマッピング

ここでは，データの要素の値によって，出力先の要素の値を変えてマッピングする方法について説明します。

これまでの例では，値を持つ要素「単位」は，入力側の要素「単位」を出力側の要素「単位」にそのまま代入するマップ式を定義しました。ここでは，メッセージの要素「単位」の値によって，出力先のメッセージの要素「単位」の値に変えてマッピングする例について説明します。固定長形式のフォーマット「FIX2」からセパレータ形式のフォーマット「SEPA2」へマッピングします。データの構造を図 4-13 に示します。

図 4-13 フォーマット「FIX2」「SEPA2」のデータの構造

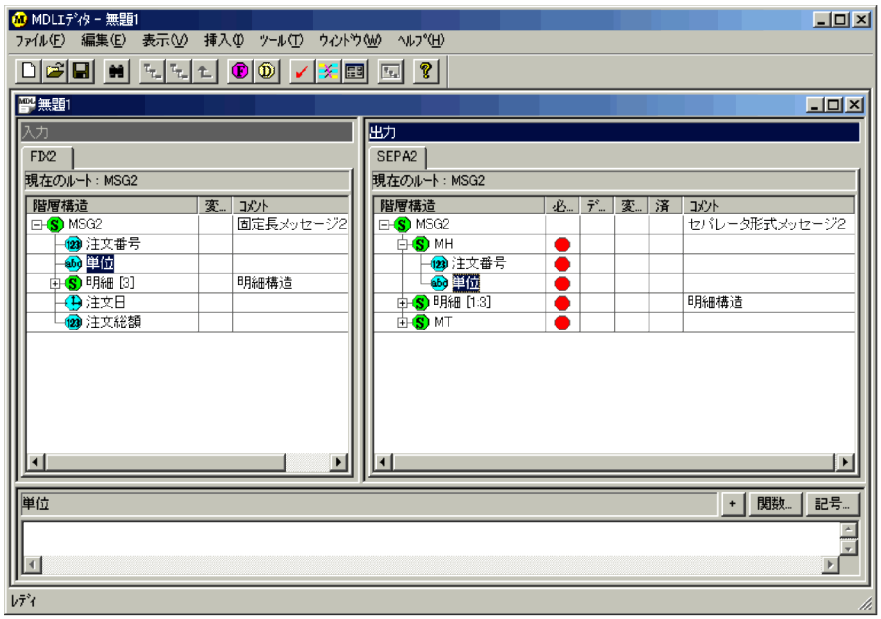


固定長形式のフォーマット「FIX2」及びセパレータ形式のフォーマット「SEPA2」の要素「単位」は，値とその意味はすべて同じです。値とその意味を次に示します。

- 値「PC」，意味「個」
- 値「KG」，意味「キログラム」
- 値「M」，意味「メートル」

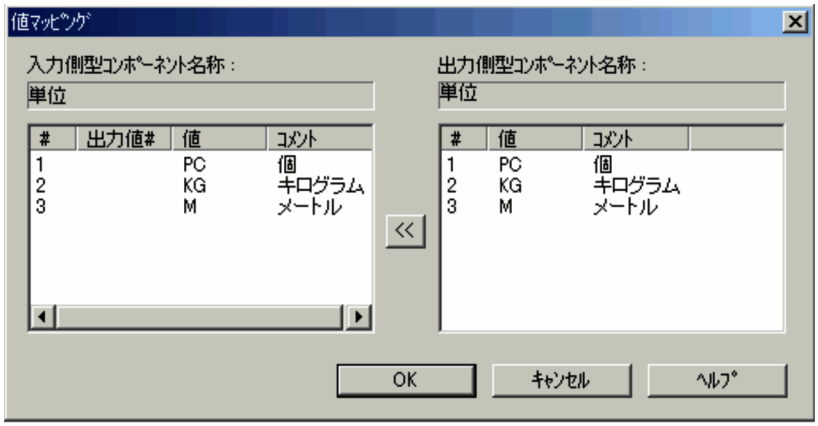
要素の値同士を対応付けるには，値をマッピングします。

1. **入力側及び出力側のコンポーネント「単位」を選択します**



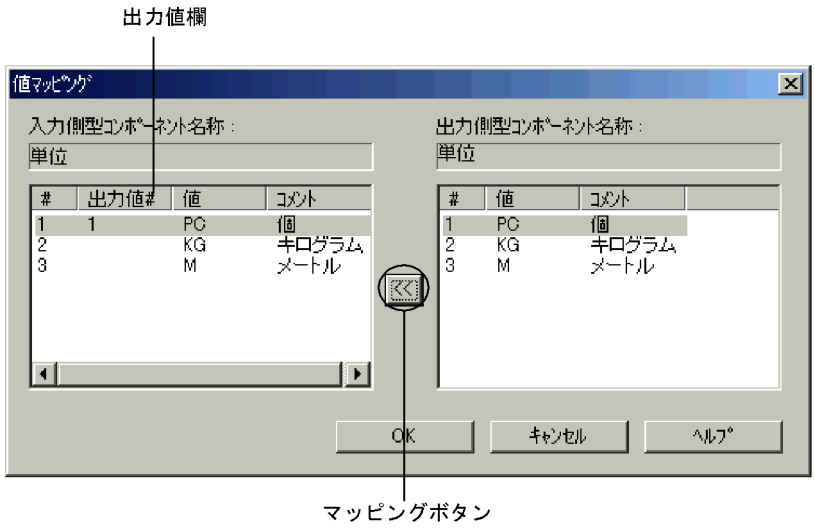
2. **［ツール］－［マップ式］－［値マッピング］を選択します**
「単位」は値を定義した要素なので，［値マッピング］ダイアログが表示されます。

図 4-14 ［値マッピング］ダイアログ



要素に定義した値が一覧表示されます。このダイアログで，入力要素のどの値を出力要素のどの値として出力するかを対応付けます。
この例ではまず，値「PC」をマッピングします。

3. **入力側の値「PC」と出力側の値「PC」を選択し，［<<］ボタンをクリックします**
値を選択する場合は「#」欄でインデクスを選択します。



入力側の一覧の出力値欄にマッピングしたことを示すインデクスが表示されます。このインデクス番号は，出力側の値のインデクスを表しています。この欄にインデクス番号が表示されていない値は，出力側の値がマッピングされていないことになります。
ここでは，出力側のインデクス 1 の値「PC」をマッピングしたので，入力側の値

「PC」の出力値欄には，インデクス「1」が表示されます。
同様に，入力側の値「KG」と出力側の値「KG」，入力側の値「M」と出力側の値「M」をマッピングします。
値のマッピングは，入力側の値すべてがマッピングされていないとエラーになるので，必ず各値をマッピングしてください。また，この例では値の内容が一致しているため，同じ値同士をマッピングしましたが，異なる値同士でもマッピングできます。ただし，複数の入力要素の値は一つの出力要素の値へマッピングできても，一つの入力要素の値を複数の出力要素の値へマッピングできません。また，入力側又は出力側のどちらかの値をダブルクリックすると，相手側の対応する値が選択されます。

4. **値のマッピングが済んだら，[OK] ボタンをクリックします**
［値マッピング］ダイアログが閉じると，マップビューには値のマッピング結果が表示されます。

```
= VALUEMAP(FIX2@MSG2@単位,#,1,1,2,2,3,3);
```

VALUEMAP 関数は，要素の値同士をマッピングする関数です。VALUEMAP 関数の引数について説明します。

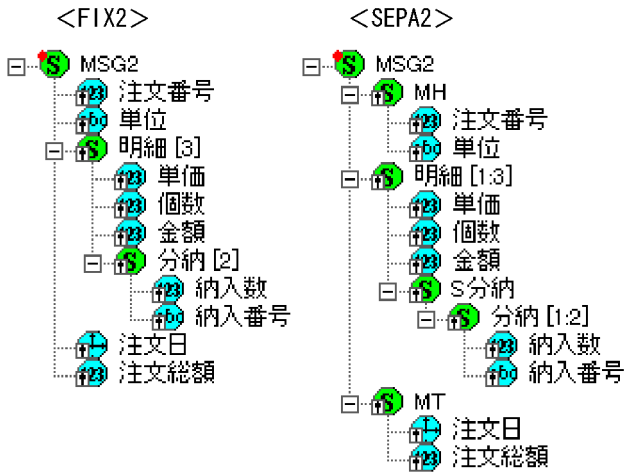
- 第 1 引数
  入力側の要素名
- 第 2 引数
  出力側の要素名（通常はマップ式を定義した自コンポーネント「#」を表します）
- 第 3 引数以降
  入力値と出力値のインデクスを対応する組ごとに記述したもの

以上で値同士のマッピングが定義できました。

4.3.4 条件式と NULL を使用したマッピング

ここでは，コンポーネントに対する条件を定義する条件式を使用したマップ式の記述について説明します。値のマッピングと同様，この例でも固定長形式のフォーマット「FIX2」とセパレータ形式のフォーマット「SEPA2」をマッピングします。データの構造を図 4-15 に示します。

図 4-15　フォーマット「FIX2」「SEPA2」のデータの構造



この例では，構造「分納」の子コンポーネント「納入数」に次の二つの規則があると想定して，マップ式を記述します。

- 要素「個数」の値が 50 を超える値 n の場合は，1 番目の「納入数」の値は 50，2 番目の「納入数」の値は n から 50 を減算した残りとする
- 「個数」の値が 50 以内の場合は，構造「分納」は 1 番目だけ出力する

この規則では，1 番目の要素「納入数」の値は最大 50 であり，要素「個数」の値でもあります。2 番目の要素「納入数」は，1 番目の要素の残りを値として出力するか又は出力されません。以上の規則に従って，マップ式を記述します。

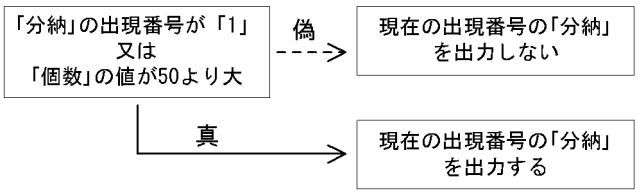
まず，「個数」の値によって 2 番目の「分納」を出力するかしないかを，「分納」に対してマップ式で記述します。ここでは，IF 関数を使用して条件式を記述します。IF 関数には引数が三つあります。IF 関数の引数に記述する内容について次に説明します。

- 第 1 引数＝条件式
- 第 2 引数＝条件式が真のときに実行される式や代入される値
- 第 3 引数＝条件式が偽のときに実行される式や代入される値

「分納」に対するマップ式を次に示します。

```
= IF((INDEX(0) == 1 | ¥2@個数 > 50), $, NULL);
```

「¥n」は式を定義するコンポーネントから n 階層上のコンポーネントを表すので，「¥2」は「分納」から 2 階層上のコンポーネント，つまり出現中の「明細」を示します。「$」は親コンポーネントのマップ式を引き継ぐための指定で，ここではコンポーネントの出力を意味します。「NULL」はコンポーネントを出力しない指定です。

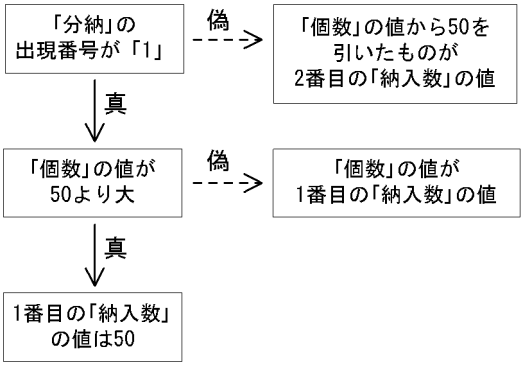


「分納」の出現番号と構造「明細」の子コンポーネント「個数」の値を条件とし，「分納」を出力するかしないかを定義しています。

次に，「納入数」に対するマップ式を示します。

```
= IF (INDEX(1) == 1, IF(¥3@個数 > 50, 50, ¥3@個数), ¥3@個数 - 50);
```

上記マップ式では，IF 関数を二つ使用しています。「¥3」は「納入数」から 3 階層上のコンポーネント，つまり出現中の「明細」を示します。

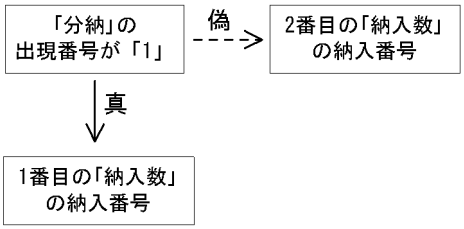


「分納」の出現番号と「個数」の値を条件とし，「納入数」の値を定義しています。

次に，「納入番号」に対するマップ式を次に示します。

```
= IF(INDEX(1) == 1, "FIRST  ", "SECOND ");
```

「FIRST」及び「SECOND」は，出現番号を表すために仮に使用した文字列です。



「分納」の出現番号を条件とし，何番目の「納入数」に対応する納入番号として出力するかを定義しています。

以上で，条件式に NULL を使用したマップ式が定義できました。なお，親のマップ式を引き継ぐ記述法「$」をマップ式で使用する場合は，親階層以上のコンポーネントにマップ式を定義しておく必要があります。また，必ず出現しなければならないコンポーネントに対して「NULL」のマップ式を記述すると，データを変換するときにエラーとなるので注意してください。

各コンポーネントのマップ式を表 4-6 に示します。

表 4-6　各コンポーネントのマップ式（条件式と NULL を使用したマッピング例）

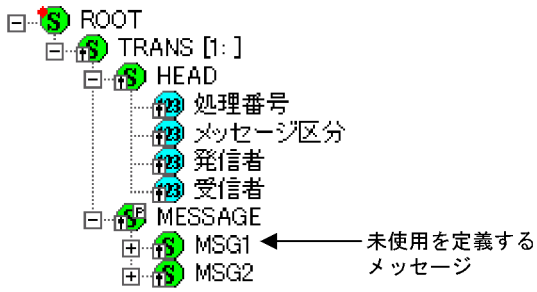
構造／要素			マップ式
MH	注文番号		= FIX2@MSG2@ 注文番号 ;
	単位		= VALUEMAP(FIX2@MSG2@ 単位 , #, 1, 1, 2, 2, 3, 3) ;
明細			= FIX2@MSG2@ 明細 [INDEX(0)];
	単価		= $@ 単価 ;
	個数		= $@ 個数 ;
	金額		= $@ 金額 ;
	S 分納		なし
		分納	= IF((INDEX(0) == 1 | ¥2@ 個数 > 50), $, NULL);
		納入数	= IF (INDEX(1) == 1, IF(¥3@ 個数 > 50, 50, ¥3@ 個数), ¥3@ 個数 - 50);
		納入番号	= IF(INDEX(1) == 1, "FIRST ", "SECOND ");
MT	注文日		=CURRENTDATE( );
	注文総額		=SUM(SEPA2@MSG2@ 明細 [*]@ 金額);

4.3.5　選択構造を持つフォーマットのマッピング

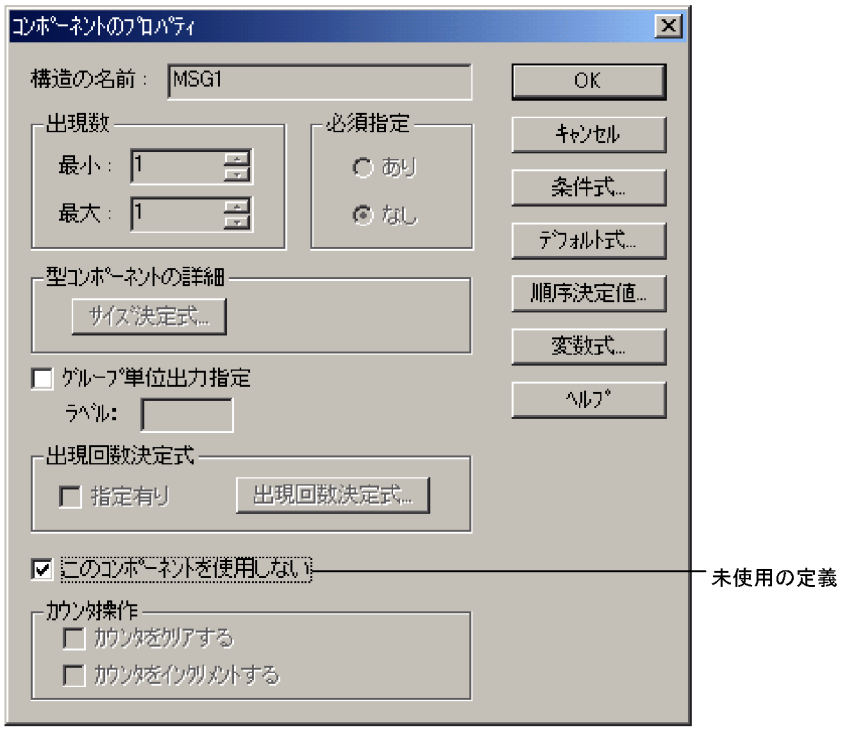
3 章のフォーマット定義例で作成したフォーマット「FIX」「LT」「SEPA」は，2 種類のメッセージを選択構造の下に配置し，ヘッダ部の値によってメッセージが選択できるようにしました。フォーマット「FIX」「LT」「SEPA」のような選択構造の場合，特別な指定は必要ありません。しかし，メッセージ数が多く，直接メッセージを選択できる条件がない場合，トランスレータは，選択構造の子コンポーネントに対して，データと適合するかどうかを順番に調べていきます。このため，メッセージ数が増えるほど，変換の効率が悪くなります。

そこで，あらかじめ入出力データとして現れないメッセージが分かれば，MDL エディタで「このメッセージ構造は出現しない」を指定できます。この指定をすると，FDL ファイルで定義されていたメッセージ構造は，MDL ファイル上では定義されていないものとして扱われます。したがって，データと適合しないメッセージを調べる必要がなくなり，トランスレータの処理速度が向上します。

フォーマット「FIX」を使用し，メッセージの未使用の定義について説明します。ここでは，メッセージ「MSG1」に対して未使用の定義をします。



メッセージ「MSG1」の［コンポーネントのプロパティ］ダイアログで，「このコンポーネントを使用しない」を選択します。これで未使用の定義が完成しました。



4.4　XML データのマッピング例

この節では，XML 形式のフォーマットの MDL ファイルを定義する方法について説明します。

ここでは，可変長形式のフォーマットから，XML 形式のフォーマットへマッピングする例を使用します。

可変長形式のフォーマットとして「RLOCAL」，XML 形式のフォーマットとして「RXML」を使用します。「RLOCAL」と「RXML」のデータ構造を次に示します。

図 4-16　フォーマット「RLOCAL」「RXML」のデータ構造

挿入するフォーマットの読み込み方法については，「4.2.1　フォーマットの挿入」を参照してください。

次に，出力フォーマット「RXML」の読み込み方法について説明します。

1. **出力フォーマットビューを選択し，［挿入］－［DTD フォーマット］を選択します**
   ［DTD フォーマット挿入：出力］ダイアログが表示されます。なお，Windows エクスプローラを開き，挿入したい DTD ファイルをフォーマットビューにドラッグ & ドロップして挿入することもできます。

図 4-17 ［DTD フォーマット挿入：出力］ダイアログ

出力側のフォーマットの DTD ファイル名，フォーマット名，ルート要素名，出力データのファイル名を指定するダイアログです。

2. **出力側のフォーマットの DTD ファイル名を指定します**
   DTD ファイル名は直接入力できますが，［参照］ボタンをクリックすると，DTD ファイルを参照して指定できます。
   ここでは，出力フォーマットに「ryohi.dtd」を指定します。

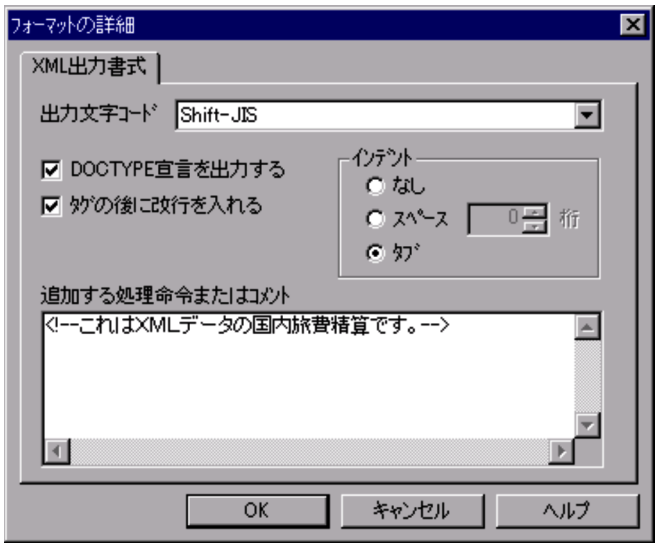
3. **フォーマット名を指定します**
   フォーマット名は，DTD ファイル名を指定すると，自動的に生成されます。ここで，任意の名称に変更できます。

4. **ルート要素名を指定します**
   DTD ファイル中にルートが一つの場合は，ルート要素名が自動的に挿入されます。ルートが複数ある場合は，DTD ファイルを指定したときに表示されるか，［選択］ボタンを押して表示される［ルート要素選択］ダイアログから，ルート要素名を選択してください。

5. **出力データファイル名を指定します**
   出力データファイル名は，データを変換するときのコマンドの引数で指定できるので，このダイアログでは未指定でもかまいません。なお，出力データに入力データと同じファイル名を指定するとエラーになるので，異なるファイル名を指定してください。
   次に，出力する XML 文書の書式を指定します。

6. **［DTD フォーマット挿入：出力］ダイアログの［詳細］ボタンを押します**
   ［フォーマットの詳細］ダイアログが表示されます。

図 4-18 ［フォーマットの詳細］ダイアログ



ここでは，次のように指定します。

- 出力文字コード：Shift-JIS
- DOCTYPE 宣言を出力する
- タグの後に改行を入れる
- インデント：タブ
- 追加する処理命令又はコメント：これは XML データの国内旅費精算です。※

  注※　この項目は，テキストで記述しますが，XML 1.0 の規則に従って，記述してください。

次のように［フォーマットの詳細］ダイアログを表示する方法もあります。

- 出力フォーマットを選択した状態で，［ツール］－［プロパティ］－［フォーマット］を選択する

  ［フォーマットのプロパティ］ダイアログが表示されます。
- ［フォーマットのプロパティ］ダイアログの［詳細］ボタンを押す※

  ［フォーマットのプロパティ］ダイアログが表示されます。

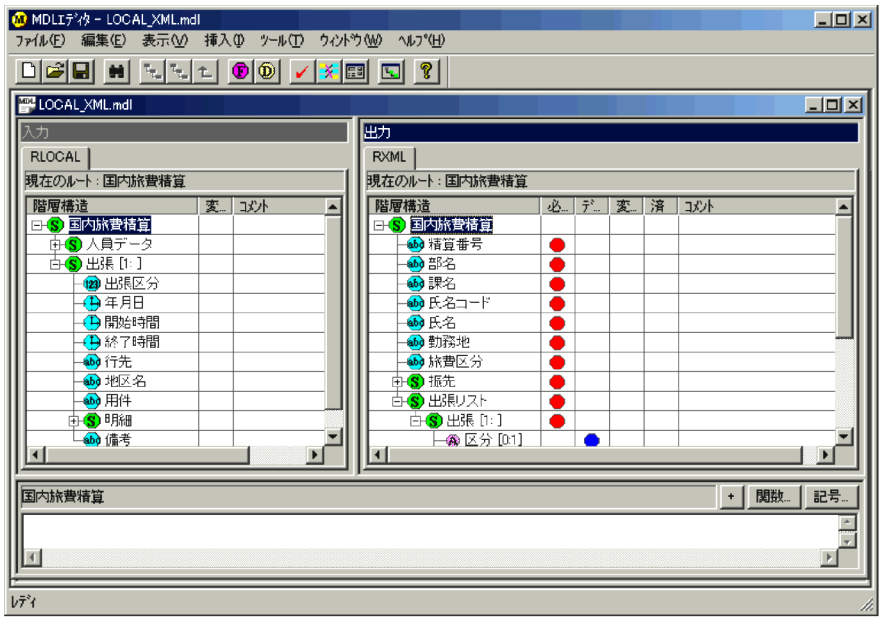
  注※　入力フォーマットを選択していた場合は，［詳細］ボタンは非活性となります。DTD フォーマットの詳細は，出力フォーマットに対してだけ有効です。

7. **［フォーマットの詳細］ダイアログの［OK］ボタンをクリックします**

   ［フォーマットの詳細］ダイアログが閉じて，［DTD フォーマット挿入：出力］ダイアログに戻ります。

8. **［DTD フォーマット挿入：出力］ダイアログで［OK］ボタンをクリックします**

   出力フォーマットビューに，選択したフォーマットが挿入されます。



指定した DTD フォーマットに再帰構造が含まれている場合は，［展開ネスト数指定］ダイアログが表示されるので，再帰構造の展開ネスト数を指定してください。
次に，入力側と出力側のコンポーネントを対応付け，出力側のコンポーネントに対してマップ式を定義します。

4.4.1　マップ式の定義（XML データの場合）

XML 要素に加え，XML 属性もコンポーネントとして扱われます。通常の型コンポーネントと同様に，ドラッグ＆ドロップでマップ式を定義して出力する値を決めたり，入力された XML 属性の値を通常の型コンポーネントの値としてマッピングしたりできます。

マッピングの操作方法については，「4.2.2　マップ式の定義」を参照してください。また，MDL ファイルの検証と保存方法については，「4.2.3　MDL の検証」及び「4.2.4 MDL ファイルの保存」を参照してください。

XML データのマッピング例の各コンポーネントのマップ式を次に示します。

表 4-7　各コンポーネントのマップ式（XML データの場合）

要素	マップ式
国内旅費精算	—
精算番号	= RLOCAL@ 国内旅費精算 @ 人員データ @ 精算番号 ;
部名	= RLOCAL@ 国内旅費精算 @ 人員データ @ 部名 ;
課名	= RLOCAL@ 国内旅費精算 @ 人員データ @ 課名 ;
氏名コード	= RLOCAL@ 国内旅費精算 @ 人員データ @ 氏名コード ;

要素	マップ式
氏名	= RLOCAL@ 国内旅費精算 @ 人員データ @ 氏名 ;
勤務地	= RLOCAL@ 国内旅費精算 @ 人員データ @ 勤務地 ;
旅費区分	= RLOCAL@ 国内旅費精算 @ 人員データ @ 旅費区分 ;
振先	―
振区	= RLOCAL@ 国内旅費精算 @ 人員データ @ 振区 ;
振替 NO	= RLOCAL@ 国内旅費精算 @ 人員データ @ 振替 NO;
出張リスト	―
出張 [1:]	= RLOCAL@ 国内旅費精算 @ 出張 [INDEX(0)];
区分	= $@ 出張区分 ;
年月日	= $@ 年月日 ;
時間	―
開始時間	= $@ 開始時間 ;
終了時間	= $@ 終了時間 ;
行先	= $@ 行先 ;
地区名	= $@ 地区名 ;
用件	= $@ 用件 ;
明細	―
電車	= $@ 明細 @ 電車 ;
バス	= $@ 明細 @ バス ;
タクシー	= $@ 明細 @ タクシー ;
日当	= $@ 明細 @ 日当 ;
合計	= $@ 明細 @ 合計 ;
備考	= $@ 備考 ;

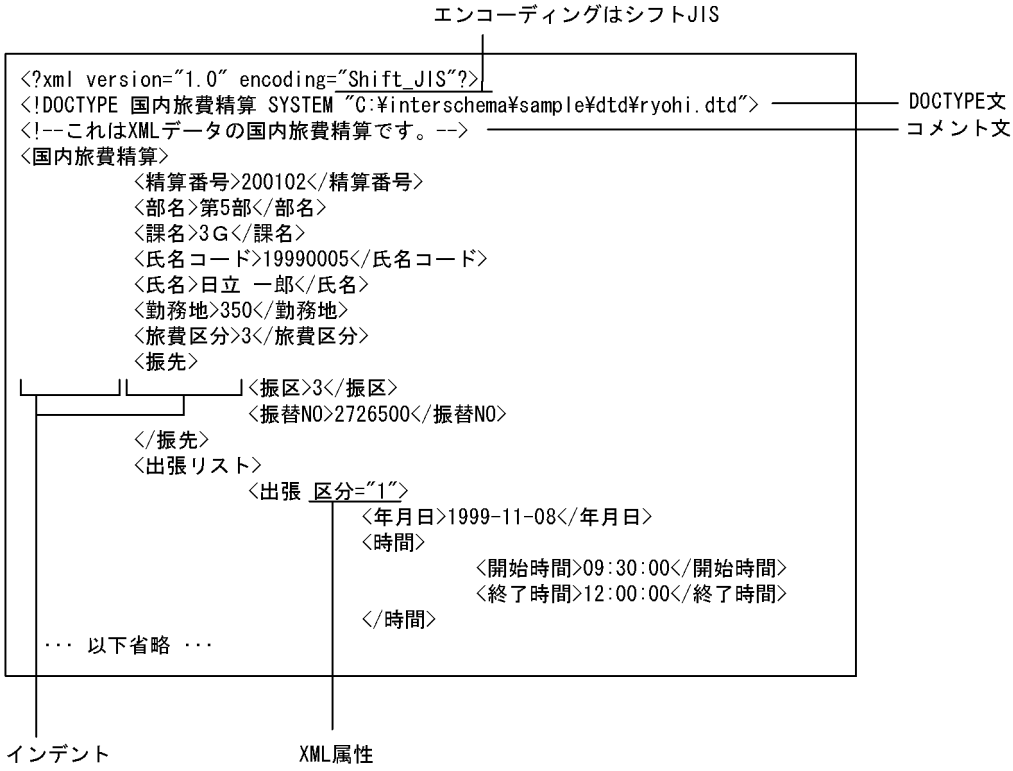
（凡例）

―：該当しません。

4.4.2 変換の結果（XML データの場合）

XML データのマッピング例に従って変換した結果を次に示します。

図 4-19　変換の結果


エンコーディングはシフトJIS
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE 国内旅費精算 SYSTEM "C:¥interschema¥sample¥dtd¥ryohi.dtd">
DOCTYPE文
<!--これはXMLデータの国内旅費精算です。-->
コメント文
<国内旅費精算>
<精算番号>200102</精算番号>
<部名>第5部</部名>
<課名>3G</課名>
<氏名コード>19990005</氏名コード>
<氏名>日立 一郎</氏名>
<勤務地>350</勤務地>
<旅費区分>3</旅費区分>
<振先>
<振区>3</振区>
<振替NO>2726500</振替NO>
</振先>
<出張リスト>
<出張 区分="1">
<年月日>1999-11-08</年月日>
<時間>
<開始時間>09:30:00</開始時間>
<終了時間>12:00:00</終了時間>
</時間>
・・・ 以下省略 ・・・
インデント
XML属性

4.5 MDL ファイル作成時の注意事項

この節では，MDL ファイルを作成するときの注意事項について説明します。

(1) フォーマット挿入時の検証

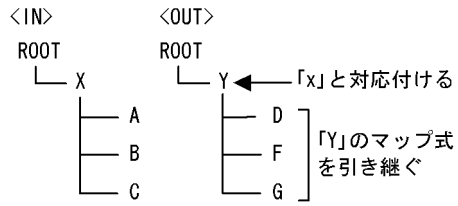
新規にフォーマットを挿入する場合，セパレータが正しく設定されているかなどが自動的に検証されます。この検証でエラーが発生した場合は，FDL エディタでエラーを修正後，再度フォーマットを挿入してください。

(2) マップ式

出力側のコンポーネントに対しては，マップ式を記述する必要があります。構造を表すコンポーネントに対しては，要素を代入するマップ式は記述できません。

(a) 定義方法

マップ式は上位の構造から順番に記述するようにしてください。また，子コンポーネントは，親コンポーネントのマップ式を引き継ぐための記述法「$」を使用して記述するようにしてください。



（凡例）　[ ]内は出現回数です。

まず，入出力フォーマット間の最上位の構造同士を対応付けるマップ式を記述します。次に，その子コンポーネント間で対応するもの同士の式に，先頭に「$」を付けて，親コンポーネントのマップ式を引き継ぐ形にします。同様に，子コンポーネントに対しては親コンポーネントのマップ式を引き継ぐ形式で記述していってください。

(b) 省略できるコンポーネントのマップ式

親構造の最小出現回数に 0 が指定されていて省略できる場合，その構造にマップ式が記述されていなかったときは，子コンポーネントにマップ式が記述されていても出力されません。構造全体が省略できる場合に，その構造の内容を出力したいときは，構造のコンポーネントにマップ式を記述してください。

(c) 出現回数が不定のコンポーネントのマップ式

出現回数が不定のコンポーネントに対してもマップ式を記述する必要があります。マップ式を記述する対象が無限に出現できるコンポーネントの場合，INDEX 関数を使用し，入力側の構造コンポーネントとの対応を表すマップ式を記述する必要があります。

(d) 評価を遅延させるマップ式

マップ式を評価する時点で，マップ式の中に値が決まっていないコンポーネントが含まれている場合は，式の評価結果はエラーになってしまいます。この場合，マップ式に出てくるコンポーネントの値が決まるまで，マップ式の評価を遅延させる「%delay」を指定したマップ式を記述する必要があります。

(e) EMPTY 要素のマップ式

DTD フォーマットの EMPTY 要素に対するマップ式は，この要素が 1 回だけ出現する場合には必要ありません。省略できない，可変回出現する要素の場合は，通常のコンポーネントと同様のマップ式の記述が必要です。

(f) 再帰構造の末端にあるダミー要素のマップ式

DTD フォーマットの再帰構造の末端にあるダミー要素（EMPTY 要素）に対しては，マップ式を記述できません。また，再帰構造の末端にあるダミー要素の構造パス名を，ほかのコンポーネントのマップ式に記述できません。

(3) グループ単位出力指定

一度に変換する入力データのサイズが大規模な場合，意味のあるデータの 1 単位ごとに出力するよう指定（グループ単位出力指定）できます。グループ単位出力指定がない場合，変換に必要な空きメモリ容量が不足してしまい，変換できないことがあります。また，空きメモリ容量が足りていても，1 か所以上エラーが発生すると変換全体が失敗することになり，変換結果として何も出力されません。

グループ単位出力指定は，入力側と出力側のフォーマットを対応させて，それぞれでグループ単位出力を指定します。上位にある無限の出現回数のコンポーネントが，グループ単位出力指定の設定対象です。なお，グループ単位出力の指定をした場合，グループ 1 回分の出力データは対応する入力データ 1 回分の中ですべて決まらなければならないことに注意してください。グループ単位出力指定の詳細については，「6.1.2(1) グループ単位出力指定」を参照してください。

(4) FDL ファイルを変更した場合

MDL ファイル作成後に FDL ファイルのフォーマットを変更した場合は，MDL ファイルから変更前のフォーマットを削除し，変更後のフォーマットを読み込んで MDL ファイルを作成し直す必要があります。

入力側のフォーマットを変更した場合は，記述済のマップ式は保存されているので，コンポーネント名やツリー構造を変更していなければ，そのまま使用できます。しかし，出力側のフォーマットを変更した場合は，既存のマップ式は削除されているので，再マッピングする作業が発生します。

このような場合に，Interschema で提供する MDL エディタのマージコマンド，又はマージツールを使用すると，出力側のフォーマットを変更した場合でも，既存のマップ

式を保持しながらフォーマットの変更内容を MDL に反映するため，再マッピングの作業を最小限に抑えることができます。マージツールについては，「7. ユティリティ」を参照してください。

(5) MDL エディタでの追加・変更

次の式や値は，MDL エディタ上で追加・変更できます。

- 順序決定式
- 順序決定値
- 条件式
- デフォルト式
- サイズ決定式
- 出現回数決定式
- 変数式

ただし，FDL エディタ上での定義とは異なり，式や値を直接指定したコンポーネントに対してだけ設定が有効になります。同じ構造やコンポーネントが，別の箇所で使用されている場合，その式や値には反映されません。また，サイズ決定式及び出現回数決定式は，コンポーネントのサイズや出現回数に動的に決定する指定がある場合だけ，編集できます。これらの式が入力コンポーネントの場合は，削除できません。

5 変換のコマンド

この章では，トランスレータを使用してデータを変換する場合のコマンドについて説明します。

5.1 システム環境変数の設定（ワークステーションの OS の場合）

ワークステーションの OS で Interschema を使用するには，次のシステム環境変数を設定しておく必要があります。

- INTERSCHEMA：インストールディレクトリの設定
- SHLIB_PATH（HP-UX の場合），LIBPATH（AIX の場合），LD_LIBRARY_PATH（Solaris の場合）：共用ライブラリ検索パスの設定
- LANG：ロケールの設定

また，データ変換処理 API（Java 言語）を使用する場合は，次のシステム環境変数を設定しておく必要があります。

- CLASSPATH：クラスパスの設定

5.1.1 インストールディレクトリの設定

Interschema のインストールディレクトリは「/opt/hitachi/interschema」です。インストール後にディレクトリを変更する場合は，環境変数「INTERSCHEMA」を使用して設定します。

（例）csh（C シェル）で，環境変数「INTERSCHEMA」にディレクトリ「/users/interschema」を設定する

```
setenv INTERSCHEMA /users/interschema
```

環境変数「INTERSCHEMA」を設定していない場合は，「/opt/hitachi/interschema」が仮定されます。

5.1.2 共用ライブラリ検索パスの設定

Interschema の共用ライブラリ検索パスを設定します。OS ごとの環境変数名を次に示します。

表 5-1 共用ライブラリ検索パスの環境変数名

OS	環境変数名
HP-UX	SHLIB_PATH
AIX	LIBPATH
Solaris	LD_LIBRARY_PATH

（例）OS が HP-UX の場合，csh（C シェル）で，環境変数「SHLIB_PATH」にディレクトリ「/opt/hitachi/interschema/lib」を設定する

```
setenv SHLIB_PATH /opt/hitachi/interschema/lib
```

5.1.3 ロケールの設定

Interschema のシステム環境で使用するロケールを設定します。設定例を次に示します。

（例）csh（C シェル）で，文字コードにシフト JIS を設定する

```
setenv LANG ja_JP.SJIS
```

環境変数「 LANG」に設定できる値について説明します。

次に示す場合の文字コードは，OS のロケールに対応した文字コードになります。

- コマンドラインから入力する引数
- API の各関数をコールするときの引数
- ログファイルに出力されるメッセージ
- システム情報ファイルの内容

ただし，MDL ファイル内のフォーマット名やコンポート名には，OS のロケール種別にかかわらずシフト JIS コードが使用されます。サポート対象外のロケール種別が指定された場合，ロケール種別は「C」（英語）になります。

Interschema がサポートするロケールは，シフト JIS，日本語 EUC，及び C（英語）です。

ロケールと文字コードの対応関係を次に示します。

表 5-2 ロケールと文字コードの対応関係

OS	ロケール	文字コード
HP-UX	ja_JP.SJIS	シフト JIS
	ja_JP.eucJP	日本語 EUC
	C	C（英語）
AIX	Ja_JP	シフト JIS
	ja_JP	日本語 EUC
	C	C（英語）
Solaris	ja_JP.PCK	シフト JIS
	ja	日本語 EUC
	C	C（英語）

データ変換処理 API（C 言語）を使用してプログラムを作成する場合，環境変数「LANG」の設定を有効にするには，main 関数の冒頭で setlocale(LC_CTYPE, “ ”); 又は setlocale(LC_ALL, “ ”); を実行して，プログラムのロケールを環境変数に合わせる必要があります。setlocale を実行しない場合のロケールは，「C」（英語）になります。データ変換処理 API（C 言語）の呼び出しの前にロケールを設定している場合は，そのロケールに従います。ただし，表 5-2 に示したロケール以外の無効なロケールが設定されている場合は，「C」（英語）が仮定されます。

5.1.4 クラスパスの設定（ワークステーションの OS の場合）

Java 言語のデータ変換処理 API を使用する場合，環境変数「CLASSPATH」に，Jar ファイル「ETransJ2.jar」のパスを設定する必要があります。設定例を次に示します。

（例）csh（C シェル）で，環境変数「CLASSPATH」に，「/opt/hitachi/interschema/classes/」に格納されている「ETransJ2.jar」を設定する

```
setenv CLASSPATH /opt/hitachi/interschema/classes/ETransJ2.jar
```

5.2 システム環境変数の設定（Windows の場合）

Windows で Interschema を使用する場合，データ変換処理 API（Java 言語）を使用するためには，次のシステム環境変数を設定する必要があります。

● CLASSPATH：クラスパスの設定

● PATH：ライブラリ検索パスの設定

5.2.1 クラスパスの設定（Windows の場合）

環境変数「CLASSPATH」に，Jar ファイル「ETransJ2.jar」のパスを設定します。設定例を次に示します。

（例）環境変数「CLASSPATH」に，

「C:¥Program Files¥HITACHI¥Interschema2¥classes」に格納されている
「ETransJ2.jar」を設定する（1 行で記述します）

```
set CLASSPATH=C:¥Program
Files¥HITACHI¥Interschema2¥classes¥ETransJ2.jar
```

5.2.2 ライブラリ検索パスの設定

環境変数「PATH」に，Interschema が提供するライブラリの検索パスを設定します。設定例を次に示します。

（例）環境変数「PATH」に，「C:¥Program Files¥HITACHI¥Interschema2¥bin」を設定する

```
set PATH=C:¥Program Files¥HITACHI¥Interschema2¥bin
```

5.3 システムファイルの設定

Interschema が使用するシステムファイルは，次の三つです。

- システム情報ファイル「ettrans.ini」
- CII データタグファイル「etciitag.ini」
- EIAJ ハッシュ用ファイル「ethash.ini」

システムファイルを設定するには，トランスレータをインストールした後に，各システムファイルをテキストエディタなどで開いて設定してください。各システムファイルは，次のディレクトリ下に格納されています。

Interschema のインストールディレクトリ\bin

次に，システムファイルごとに設定内容及び設定方法について説明します。

(1) システム情報ファイル「ettrans.ini」の設定

システム情報ファイル「ettrans.ini」には，Interschema の動作条件などを設定します。システム情報ファイルの設定内容と設定方法を次に示します。

(a) ログファイルサイズの変更

Interschema のログファイルは，データの変換結果の戻り値やエラーメッセージなどのログを出力するファイルです。トランスレータは，指定された最大サイズの範囲内でログをファイルへ追加して出力します。ログファイルが最大サイズを超えた場合，古いログ情報から削除します。ログファイルの最大サイズは，システム情報ファイルで変更できます。

システム情報ファイルの記述形式について説明します。

ログファイルの最大サイズの変更は，[Log] セクションに設定します。

- Size キー
  「1」から「10,000」までの範囲で「Size」の値を変更します。単位はキロバイトです。
  初期値で設定されている最大サイズは，1,000 キロバイトです。

システム情報ファイル「ettrans.ini」の記述例を次に示します。

```
;ettrans.ini

[Log]
Size = 1000
   ・
   ・
```

(b) ユーザ組み込み関数の定義

Interschema が標準で提供している関数以外に，ユーザが独自に定義した関数（ユーザ

組み込み関数）を使用するためには，システム情報ファイルにユーザ組み込み関数の情報を定義します。詳細については，「11.3　システム情報ファイル「ettrans.ini」の定義」を参照してください。

(c) 実行時オプションの定義

Interschema の実行時オプションは，固定的に使用するか使用しないかをあらかじめシステム情報ファイルで定義できます。詳細については，「5.4.3　システム情報ファイル「ettrans.ini」での実行時オプションの定義」を参照してください。

(d) 出力エリアサイズの初期値の指定

API によるメモリデータの出力変換での，システムが自動的に確保する出力エリアサイズの初期値を指定します。このセクションでの指定は，API 引数のアドレスリストでの指定がなかった場合だけ適用されます。アドレスリストとこのセクションにも指定がない場合は，初期値が適用されます。関連する API は，ETtrans2Exec です。

出力エリアサイズの初期値は，[MemoryData] セクションに設定します。

- InitSize キー
  出力エリア初期割り当てサイズ（バイト）
  初期値 =1,024
- IncrementSize キー
  出力エリア割り当て増分サイズ（バイト）
  初期値 =1,024

システム情報ファイル「ettrans.ini」の記述例を次に示します。

```
;ettrans.ini
   ・
   ・
[MemoryData]
InitSize = 1024
IncrementSize = 1024
```

(2) CII データタグファイル「etciitag.ini」の設定

CII データタグファイル「etciitag.ini」は，Interschema の実行時オプション「-CIIT」を指定して CII 標準データを出力する場合に使用します。ただし，CII3 フォーマットの場合，このファイルは必要ありません。

Interschema の実行時オプション「-CIIT」を指定して CII 標準データを出力する場合，省略の対象となる TFD 項目は出力しません。省略の対象となる TFD 項目については，「付録 G.4(4)　TFD 項目の省略」を参照してください。省略の対象となる TFD 項目は，項目の属性とデータタグ番号の対応を，CII データタグファイルに設定しておく必要があります。なお，トランスレータでは，EIAJ のバージョン 2H 版のメッセージに対応するファイルを提供していますので，2H 版の EIAJ 標準メッセージと異なるメッセージを変

換する場合だけ，TFD 項目の設定を変更してください。

CII データタグファイルの記述形式について説明します。

項目の属性ごとにデータタグ番号を指定します。

● ［X］セクション
省略の対象となる X 属性のデータタグ番号を指定します。

● ［K］セクション
省略の対象となる K 属性のデータタグ番号を指定します。

● ［9］セクション
省略の対象となる 9 属性／ N 属性／ Y 属性のデータタグ番号を指定します。

● ［B］セクション
省略の対象となる B 属性のデータタグ番号を指定します。

● データタグ番号
セクションごとに 10 進数でデータタグ番号を指定します。データタグ番号は，1 行に 1 形式だけ指定できます。データタグ番号の指定形式は次の 2 種類です。

nn
　タグ番号「nn」を意味します。

nn-mm
　タグ番号「nn，nn+1，…，mm」までを意味します（nn < mm）。

CII データタグファイル「etciitag.ini」の記述例を次に示します。

```
;etciitag.ini

[X]
2
4-10

[K]
260-280

[9]
1
3
11

[B]
```

（3） EIAJ ハッシュ用ファイル「ethash.ini」の設定

EIAJ ハッシュ用ファイル「ethash.ini」は，EIAJ 標準の方式でハッシュトータルチェックをする場合に使用します。ただし，CII3 フォーマットの場合，このファイルは必要ありません。

変換するメッセージデータが EIAJ 標準に従うと見なす場合は，EIAJ 標準の計算方式で

ハッシュトータルチェックを実施します（詳細については，「付録 G.4(3) ハッシュトータルチェック」を参照してください)。EIAJ 標準方式で値をチェックするには，EIAJ 標準メッセージで使用する数値属性と小数部桁数の対応を，EIAJ ハッシュ用ファイルに設定しておく必要があります。なお，トランスレータでは，EIAJ のバージョン 1D 版のメッセージに対応するファイルを提供していますので，1D 版の EIAJ 標準メッセージと異なるメッセージを変換する場合だけ，設定を変更してください。

EIAJ ハッシュ用ファイルの記述形式について説明します。

数値属性項目の TFD データタグ番号と小数部桁数を [EIAJ] セクションに設定します。

● ［EIAJ］セクション
EIAJ 標準メッセージで定義する項目のうち，小数部を含む項目を定義します。
キー名として項目番号（3 桁で指定）を指定します。右辺には，項目の小数部桁数を指定します。

EIAJ ハッシュ用ファイル「ethash.ini」の記述例を次に示します。

```
;ethash.ini

[EIAJ]
013=3
015=3
   ・
   ・
   ・
190=3
```

5.4 トランスレータでのデータ変換

データファイルの変換を実行するトランスレータのコマンド「ettrans」について説明します。

5.4.1 ettrans コマンド

ettrans コマンドは，次のディレクトリ下に格納されています。

Interschema のインストールディレクトリ /bin

(1) 機能

指定された MDL ファイル内の定義内容に従って，入力データを出力データへ変換します。

コマンド名とオプションを逐次，入力指定する形式と，指定したオプションを定義したパラメタファイルを指定する形式があります。

パラメタファイルとは，ettrans コマンドの実行時に指定できるオプションを記述したテキストファイルです。トランスレータの引数をパラメタファイルに記述しておけば，定常的に指定するオプションを定義しておけるので，コマンド入力の省力化を図ることができます。

(2) 形式

形式 1

```
ettrans
     MDLファイル名
     [-F {フォーマット名　入力データファイル ｜ 出力データファイル名}...]
     [-FN フォーマット名...]
     [-E ログファイル名]
     [実行時オプション]
```

形式 2

```
ettrans -file  パラメタファイル名
```

形式 3

```
ettrans -prop  FDLファイル名 ｜ MDLファイル名
```

（3）引数

（a）形式 1 で指定できる引数

MDL ファイル名

変換したいデータの変換情報を定義した MDL ファイルを指定します。

-F

MDL ファイル内で定義された入力データ，出力データのファイル名を無視して，引数に指定したファイル名を変換するオプションです。
MDL ファイルに定義されていないファイルを変換したい場合，このオプションを指定してください。また，MDL ファイル内で入力データ，出力データのファイル名の設定を省略した場合にも，このオプションを指定してください。

フォーマット名

変換対象となるデータファイルに対応するフォーマット名を指定します。MDL ファイル内で定義されているフォーマット名を指定してください。

入出力データファイル名

指定したフォーマット名に対応する入力データ又は出力データのファイル名を指定します。

-FN

MDL で定義されたフォーマットのうち，指定したフォーマットのデータだけを変換するオプションです。このオプションを指定した場合，指定されなかったフォーマットは，MDL 中にないものとして扱われ，対応するデータは変換されません。

フォーマット名

変換対象となるデータファイルに対応するフォーマット名を指定します。
フォーマット名は，MDL ファイル内で定義されているフォーマット名を指定してください。

-E ログファイル名

Interschema のエラー情報などを出力するログファイル名を指定します。この指定を省略した場合は，デフォルトのファイル名「errlog.txt」でログファイルを出力します。ログファイルの格納先は，「*Interschema のインストールディレクトリ*\log」下です。ただし，システム情報ファイルの [Option] セクションにログの出力を抑止する設定（NE=1）がある場合，ログファイルは出力されません。ログファイルの出力を抑止する場合は，「5.4.3　システム情報ファイル「ettrans.ini」での実行時オプションの定義」を参照してください。

実行時オプション

複数のオプションを指定する場合は，オプション間をスペースで区切ってください。
実行時オプションを次に示します。

-CIIE 出力ファイル名

CII 標準形式の入力データを変換する場合，出力データと同時に CII 標準のエ

ラー情報メッセージを出力したいときに指定します。

-CIIR **出力ファイル名**

CII 標準形式の入力データを変換する場合，CII 標準の受信確認メッセージを出力したいときに指定します。出力データと同時に出力されます。

-CIIT

CII フォーマットのデータを出力する場合，省略の対象となる TFD 項目の出力を抑止したいときに指定します。
省略の対象となる TFD 項目については，「付録 G.4(4)　TFD 項目の省略」を参照してください。省略の対象となる TFD 項目は，属性とデータタグ番号の対応を，CII データタグファイル「etciitag.ini」で設定しておく必要があります。
CII データタグファイルの設定については，「5.3(2)　CII データタグファイル「etciitag.ini」の設定」を参照してください。
CII3 フォーマット出力時に指定した場合は，縮小モードで出力できるマルチ明細が，番号なしマルチ明細として出力されます。CII3 フォーマットでは，「-CIIT」オプションを指定していなくても，省略できる TFD は常に省略されます。

-CSV

CSV フォーマットのデータを出力する場合，CSV データ項目をダブルクォーテーション「"」で囲みたいときに指定します。このオプションは，出力するデータが CSV フォーマットの場合だけ有効です。CSV データ項目と見なされないデータ（バイト列など）は，ダブルクォーテーション「"」で囲まれません。

-DEFMAP

コンポーネントにデフォルト式が設定されていて，コンポーネントのマップ式の評価がエラーになった場合，デフォルト式によって値（デフォルト値）を出力したいときに指定します。デフォルト値が適用される対象項目は次のとおりです。

- 要素のマップ式の評価がエラーになった場合のその要素
- 構造のマップ式の評価がエラーになった場合の子コンポーネントすべて

デフォルト値は，該当するコンポーネントの最小出現回数分だけ出力されます。ただし，NULL マップ式を記述した結果，コンポーネントの出現回数が不足する場合はエラーになり，デフォルト値は出力されません。
なお，マップ式の評価結果の代わりにデフォルト式の値が出力されても，ログファイルにメッセージは出力されません。

-DSEPA

セパレータを持つコンポーネントデータ（構造データ）を出力する場合に，不要と見なされるセパレータを出力したくないときに指定します。
不要なセパレータとは，構造コンポーネントの中間区切り文字で，自分自身より後に構造の子コンポーネントデータがないものを指します。
ある構造のセパレータとして，開始文字「SEG」，中間区切り文字（位置は各要

素の前)「+」，終了文字「'」が定義されている場合は次のように出力されます。
ここで，「e1」及び「e3」は要素データを表します。
-DSEPA 指定なし：SEG+e1++e3+++'
-DSEPA 指定あり：SEG+e1++e3'
なお，出力対象フォーマットが EDIFACT シンタックスに従う場合，このオプションの指定がなくても不要なセパレータは出力しません。

-DTM

日付，時刻，日付時刻型の各パート値に対して範囲を検証したい場合に指定します。各パート値の範囲は「6.2.4　日付時刻」を参照してください。ただし，ss パート (秒) の小数部については，検証されません。
なお，このオプションで日付，時刻としての正当性は検証されないので注意してください。

-EIAJHASH

出力変換するメッセージデータが EIAJ 標準に従うと見なす場合，EIAJ 標準の方式でハッシュトータル値を計算したいときに指定します。このオプションは，出力するデータが CII 標準又は EIAJ 標準に従っている場合だけ有効です。
EIAJ 標準に従うと見なす条件は，メッセージグループ・ヘッダ（MGH）の BPID 版の値が「1x」（x は任意）となっていることです。EIAJ 標準に従ったハッシュトータル値を計算する場合，数値属性と小数部桁数の対応を，EIAJ ハッシュ用ファイル「ethash.ini」で設定しておく必要があります。EIAJ ハッシュ用ファイルの設定については，「5.3(3)　EIAJ ハッシュ用ファイル「ethash.ini」の設定」を参照してください。ただし，出力するデータが CII3 フォーマットの場合は，EIAJ ハッシュ用ファイル「ethash.ini」を設定する必要はありません。

-ESEPA

EDIFACT フォーマットのデータを変換する場合，UNA セグメントで定義されたセパレータ，又はデフォルトのセパレータを，一律に判定するオプションです。このオプションは，従来の処理では望む変換結果が得られない場合に指定してください。
なお，このオプションを指定する場合，変換対象となる入力又は出力 EDIFACT フォーマットデータ内に，UNA セグメントが複数個あってはいけません。1 回の変換では，1 交換単位分の変換だけとしてください。

-FDS

ゾーン形式数値の EBCDIK 文字コードでの正符号として，0xF を出力したい場合に指定します。

-IERR

変換データ中に不正文字コードがあった場合，エラーとして処理したいときに指定します。
このオプションを指定しない場合は，MDL の指定に従います。

-RELVL

ettrans コマンドの戻り値としてエラーレベルを返します。エラーレベルと戻り値の対応を表 5-3 に示します。なお，ログファイルに出力する戻り値，及び各 API の戻り値は，「5.4.2　ettrans コマンドの戻り値」の値を返します。
このオプションは，API のパラメタとして指定できません。

表 5-3　エラーレベルと戻り値の対応

エラーレベル	対応する戻り値	意味
0x00	0x00ZZZZZZ	インフォメーション
0x01	0x01ZZZZZZ	ワーニング
0x02	0x02ZZZZZZ	エラー
0x04	0x04ZZZZZZ	システムエラー

（凡例）
ZZZZZZ は任意の数字です。

-SERR

変換データ中にサイズエラー（小数部を持つ実数値を整数型要素に代入する場合なども含みます）があった場合，エラーとして処理したいときに指定します。
このオプションを指定しない場合は，MDL の指定に従います。

-SNCODE

Sea-NACCS で規定する文字集合による文字コードチェックを実施します。
このオプションは，Sea-NACCS EDI 電文又は Sea-NACCS EDIFACT 電文の出力変換の場合だけ有効です。
なお，Interschema 03-00 までの Sea-NACCS 電文変換では，常に文字コードチェックを実施しています。旧バージョンと同じ変換をしたい場合は，このオプションを指定してください。

-XDOC **外部サブセット** URL

DOCTYPE 宣言を出力する場合に指定します。
このオプションを指定した場合，MDL ファイルでの指定に関係なく，出力 XML 文書に指定外部サブセット URL で DOCTYPE 宣言を出力します。

-XND

XML 出力変換時に，DOM を作成しない場合に指定します。
このオプションを指定した場合，メモリの使用量を抑制できます。なお，エラーが発生した場合のデータの出力状況，及びデータの変換時間は，オプションを指定しないときと異なります。

-XVC

XML 文書構造を検証する場合に指定します。

-XWS

入力する XML 文書の white space 削除を抑止したい場合に指定します。
このオプションが指定された場合，入力 XML データの white space は削除されないで，データの一部として扱われます。

実行時オプションは，システム情報ファイルを使って暗黙指定できます。詳細については，「5.4.3　システム情報ファイル「ettrans.ini」での実行時オプションの定義」を参照してください。

(b) 形式 2 で指定できる引数

-file **パラメタファイル名**

パラメタファイルを指定します。
パラメタファイルには，1 行に 1 オプション，又は 1 引数を記述します。したがって，-F オプションに，フォーマット名と入力データファイルや出力データファイルを 2 組指定する場合は，記述例に示すとおり 5 行で表現します。
次にパラメタファイルの記述例を示します。

```
BBB.mdl
-F
INformat
input.data
OUTformat
output.data
-IERR
-SERR
```

記述できる引数は，次のとおりです。先頭行には MDL ファイル名を指定します。
引数の詳細については，「5.4.1(3)(a)　形式 1 で指定できる引数」を参照してください。

- MDL ファイル名
- -F
- -FN
- フォーマット名
- 入出力データファイル名
- -E
- ログファイル名
- 実行時オプション

(c) 形式 3 で指定できる引数

-prop　FDL | MDL **ファイル名**

FDL ファイル又は MDL ファイルを指定します。指定したファイルのプロパティが表示されます。表示されるプロパティを次に示します。

- ユーザバージョン
- ユーザコメント
- 作成日時※
- 作成製品バージョン※

- 更新日時
- 更新製品バージョン※
- FDL ファイル又は MDL ファイルのバージョン

注※

FDL ファイル又は MDL ファイルが，Interschema バージョン 1 の FDL エディタ又は MDL エディタで作成されていた場合，作成日時，作成製品バージョン及び更新製品バージョンの情報は表示されません。

表示形式は，オプションを実行した時の LANG によって決まります。Windows 又は LANG C でプロパティを表示するオプションを実行した場合，シフト JIS 形式でプロパティが表示されます。プロパティの表示例を次に示します。

```
UserVersion=V0.1
UserComment=Test Model
SysCreatedDate=2002/10/01 10:00:00
SysCreatedPpVersion=0200
SysLastUpdatedDate=2002/10/10 12:00:00
SysLastUpdatedPpVersion=0200
SysDlVersion=16
```

(4) 実行例

ettrans コマンドの実行例を次に示します。なお，実行例 1 ～ 3 は形式 1 に従った例です。また，実行例 4 は形式 2 に，実行例 5 は形式 3 に従った例です。

実行例 1

MDL ファイル「AAA」内で指定されている入力データファイルを，出力データファイルに変換します。

```
ettrans AAA.mdl
```

実行例 2

MDL ファイル「BBB」で指定された変換情報に従って，入力データファイル「input.data」を出力データファイル「output.data」に変換します。不正文字，サイズエラーを検証するオプションを指定します。

```
ettrans BBB.mdl -F INformat input.data OUTformat output.data
     -IERR -SERR
```

実行例 3

MDL ファイル「CCC」内で指定されているフォーマットのうち，INI 及び OUT1 だけを変換します。入力データファイル及び出力データファイルは「CCC」内で定義されているものとします。

```
ettrans CCC.mdl -FN INI OUT1
```

実行例 4

指定したパラメタファイル「param.txt」の設定に従って，変換を実行します。

```
ettrans -file param.txt
```

実行例 5

指定した MDL ファイル「DDD」のプロパティ情報を表示します。

```
ettrans -prop DDD.mdl
```

(5) 注意事項

- 空白を含むファイル名や，ディレクトリ名を引数として指定する場合，項目全体をダブルクォーテーション「"」で囲んでください。ただし，コマンドの引数を記述するパラメタファイル内で指定する場合，1 行分で指定された内容が 1 項目として認識されるため，ダブルクォーテーションで囲まずに，引数名をそのまま記述してください。パラメタファイルについては，「5.4.1　ettrans コマンド」を参照してください。
- 入力データのファイル名と出力データのファイル名が同じ場合は，データ変換時にエラーになります。また，MDL ファイル内に定義されている入出力データと同じファイル名を -F オプションの引数に指定すると，データ変換時にエラーとなります。
- ワークステーションの OS の場合，ファイル名に「"」及び「¥」を含むとき，エスケープ文字「¥」を先頭に追加してください。

  例

″ → ¥″
¥ → ¥¥

- ファイル名，コマンドの 1 引数，パラメタファイルの 1 行などは，256 バイト以内で指定してください。
- エラーが発生した場合は，エラー内容をログファイルに出力します。詳細については，「付録 A.1　エラー発生時の処理」を参照してください。

5.4.2 ettrans コマンドの戻り値

トランスレータは，戻り値（整数）を返します。この変換処理の終了後に出力する戻り値によって変換結果やエラー内容などを調べることができます。戻り値の内容や注意事項を次に示します。

(1) 戻り値の内容

戻り値

0xYYZZZZZZ

表 5-4　ettrans コマンドの戻り値

エラーレベル	YY	ZZZZZZ	意味
I	00	000000	正常に終了しました。

エラーレベル	YY	ZZZZZZ	意味
		000008	テスト用のデータを入力しました。変換は正常に終了しました。
W	01	000001	不正文字コードを検出し，スペースに置き換えました。
		000002	要素サイズをオーバーしたためデータをカットしました。
		000004	エラーが発生したが，正常変換できたグループだけ出力しました。
		000008	テスト用のデータを入力しました。
		000010	ログファイルを開けなかったため，デフォルトのログファイル又は標準出力にログを出力しました。
		000020	CII 標準又は EIAJ 標準のハッシュトータル値がメッセージグループ・トレーラ（MGT）の値と一致しません。
		000040	出口関数の定義に不正があります。
		000080	入力フォーマット以降に入力データがあります。
		000100	「ettrans.ini」の定義に不正があります。
		000000	その他のワーニングエラーです。
E	02	000001	不正文字を検出しました。
		000002	要素サイズをオーバーしました。
		000004	エラーが発生したが，正常変換できたグループだけ出力しました。
		000008	テスト用のデータでエラーが発生しました。
		000000	その他のエラーです。
S	04	000001	ファイル入出力エラーです。
		000002	メモリ不足です。
		000004	出力データのサイズが出力できるメモリアドレスを超えました。
		000008	データ変換処理 API のスレッド固有情報又は MDL 情報に不正があります。
		000010	変換できない MDL ファイルです（対応ライセンスがありません）。
		000000	その他のシステムエラーです。

（凡例）

I（インフォメーション）：正常に終了しました。

W（ワーニング）：変換中にワーニングレベルのエラーが発生しましたが，データ変換は終了しました。

E（エラー）：変換中に続行できないレベルのエラーが発生し，データは最後まで変換されないまま途中で終了しました。

S（システムエラー）：トランスレータ内部で致命的エラーが発生し，データは最後まで変換されないまま途中で終了しました。

(2) 注意事項

トランスレータの戻り値の注意事項を次に示します。

- 同じレベルのエラーが複数発生した場合，それぞれの値の論理和が出力されます。例えば，不正文字コードと要素サイズオーバーの両方のワーニングが発生した場合の戻り値は，「0x01000001」と「0x01000002」の下 2 桁の論理和となる「0x01000003」が出力されます。
- 異なるレベルのエラーが複数発生した場合，原則として，論理和は取らずに上位エラーの戻り値が出力されます。エラーレベルの優先順位を次に示します。
  S：システムエラー＞ E：エラー＞ W：ワーニング＞ I：インフォメーション
  ただし，下 2 桁に「0x00000008」（テスト用データを入力した）が含まれる場合は「0x00000008」との論理和を取った値になります。
- システムレベルのエラー（0x04ZZZZZZ）が発生した場合，下 6 桁の値についてはほかの値との論理和は取りません。
- 戻り値として「0x01000008」又は「0x02000008」を出力するのは，テスト用の入力データとして，CII 標準又は UN/EDIFACT のデータを入力した場合だけです。
- 戻り値「0x04000008」は，API による変換実行時だけ出力されます。
- 実行時オプション「-RELVL」を指定した場合，ettrans コマンドは，戻り値の代わりにエラーレベルを返します。エラーレベルについては，「表 5-3　エラーレベルと戻り値の対応」を参照してください。

5.4.3 システム情報ファイル「ettrans.ini」での実行時オプションの定義

実行時オプションを暗黙的に指定したい場合，システム情報ファイル「ettrans.ini」を利用できます。暗黙的に指定する実行時オプションには「有効」，指定しないオプションには「無効」と定義します。

（1） 定義できる実行時オプション

実行時オプションに加え，「NE」が指定できます。「NE」について説明します。

「NE」はシステム情報ファイルでだけ定義できる実行時オプションです。コマンドのオプションとして直接指定できないので注意してください。

NE オプションは，ログファイルの出力を抑止したい場合に指定します。指定するモード値によって，抑止のレベルが異なります。NE オプションで指定できるモード値とその意味を次に示します。

NE に指定するモード値

0：
ログファイルの出力を抑止しません。初期値では 0 が指定されています。

1：
すべてのログファイルの出力を抑止します。NE オプションに 1 を指定した場合は，-E オプションによるログファイルの指定があってもログファイルは出力されません。

2：
グループ単位出力指定のインフォメーションメッセージ「KBET0006T-I」の出力を抑止します。そのほかのメッセージは出力されます。NE オプションに 2 が指定されている場合にエラーが発生すると，グループ単位出力指定のワーニングメッセージ「KBET0009T-W」の付加情報として出力済みのグループ数が出力されます。

3：
正常変換時のログファイルの出力を抑止します。ここでの正常変換とは，戻り値が「0x00000000」の場合です。

システム情報ファイルで有効又は無効を定義した実行時オプションについては，トランスレータ起動時，又は API 関数でのデータ変換時にそのオプションを指定しても，システム情報ファイルの定義内容が優先されます。例えば，指定した実行時オプションがシステム情報ファイルで無効と定義されている場合には，そのオプションは指定されなかったことになります。

システム情報ファイルで有効又は無効を定義していない実行時オプションについては，トランスレータ起動時，又は API 関数でのデータ変換時の指定に従います。

実行時オプションの有効又は無効は，システム情報ファイルの［Option］セクションに定義します。[Option] セクションでの記述形式を次に示します。

(2) 形式

オプション = 有効又は無効の指定

キー名としてオプション名を記述します（実行時オプションの「-」は除きます）。
右辺にはオプションの有効，無効を指定します。右辺の指定方法を次に示します。

- モード値を指定するオプションの場合
  オプションを有効とする場合は，モード値を指定します。モード値は，整数値で指定してください。
- ファイル名又は URL を指定するオプションの場合
  オプションを有効とする場合は，ファイル名又は URL を指定します。ファイル名又は URL は，255 バイト以内で指定してください。「__ET_DEFAULT__」は指定できません。
  無効とする場合は，「=」の右辺に何も指定しないでください。
- それ以外の実行時オプションの場合
  オプションを有効とする場合は「1」を指定，無効とする場合は「0」を指定します。

なお，Interschema で提供するシステム情報ファイルには，[Option] セクションの記述はありません。実行時オプションを定義する場合には，セクション名 [Option] から記述してください。

(3) 記述例

システム情報ファイル「ettrans.ini」の記述例を次に示します。

```
;ettrans.ini
   ・
   ・
[Option]
CIIE = error_info_file.cii※1
CIIR = ※1
   ・
IERR = 0※2
CIIT = 1※2
   ・
   ・
```

注※1

有効とするオプションにはファイル名（「XDOC」の場合は URL）を指定，無効とするオプションには何も指定しません。

注※2

有効とするオプションには「1」を指定，無効とするオプションには「0」を指定します。

6 定義と変換の規則

この章では，Interschema の定義と変換の規則について説明します。

6.1 定義内容と規則

トランスレータは，FDL ファイル及び MDL ファイルに記述された変換情報に従って動作するため，FDL ファイル及び MDL ファイルにトランスレータの変換情報を記述する必要があります。

FDL ファイル及び MDL ファイルで定義する内容と規則について説明します。

6.1.1 FDL での定義情報

FDL で定義する内容について説明します。

(1) フォーマット情報

フォーマット情報は変換データの構造を定義するもので，型，セパレータ，変数，構造から構成されます。また，フォーマット全体にかかわる情報を，フォーマット情報として定義します。

フォーマット情報を次に示します。

表 6-1 フォーマット情報

フォーマット情報	定義箇所	定義する内容
規格指定	フォーマット全体	フォーマットが従う EDI 標準などの情報を指定します。
データ格納形式	フォーマット全体	データの格納形式（ビッグエンディアン又はリトルエンディアン）を指定します。初期値は，ビッグエンディアンです。
不正文字コード置換指定	フォーマット情報と型※	不正な文字コードをスペースに置換するか，エラーにするかを指定します。
桁あふれ時の指定	フォーマット全体	データが最大サイズを超えたらエラーにするか，桁あふれの部分を切り捨てるかを指定します。
不要文字削除指定	フォーマットと型※	入力文字列の埋め字をデータと見なすか（OFF 指定），データから除外して扱うか（ON 指定）を指定します。
文字コード指定	フォーマット全体	フォーマット全体で適用する文字コードを指定します。 ここで指定した文字コードが，フォーマットの初期値の文字コードとなります。
サイズ系指定	フォーマットと型※	文字列型のサイズを数える方法（バイト数又は文字数）を指定します。初期値はバイト数です。

注※

フォーマットと型の両方で定義した場合は，型の定義が優先されます。フォーマットの定義は，指定をしていない型だけに適用されます。

各フォーマット情報の詳細について説明します。

(a) 規格指定

トランスレータは，規格指定に従った処理をします。CSV 形式でデータを処理したい場合は，FDL エディタでフォーマットのプロパティを設定するときに，規格指定の項目で「CSV」を選択します。指定する規格がない場合は「なし」を選択します。

ここで選択できるのは，「CSV」か「なし」だけです。その他の規格（CII や EDIFACT など）で作成した辞書のプロパティを編集する場合は，対応する規格の名称が表示されますが，規格指定を編集することはできません。

なお，XML の DTD フォーマットの指定には，MDL エディタを使用します。

(b) データ格納形式

入出力データで扱う 2 進数のバイトオーダ（ビッグエンディアン又はリトルエンディアン）を，フォーマットごとに定義します。

バイトオーダマークを持たない Unicode 文字列のエンディアンも，この指定に従います。

(c) 不正文字コード置換指定

入出力データに不正な文字コードがある場合，スペースに置換するか，エラーにするかを指定します。スペースに置換する場合，トランスレータは，ワーニングエラーを出力します。

(d) 桁あふれ時の指定

出力フォーマット中の要素にデータを代入するとき，データの値によっては，桁あふれ又はサイズオーバーとなる場合があります。このとき，エラーにするか，あふれ部分を切り捨てて出力するかを指定します。

トランスレータは，桁あふれ時の指定がある場合，桁あふれ部分を切り捨てて出力側要素に代入し，変換処理を続行します。このとき，トランスレータは，ワーニングエラーを出力します。桁あふれ切り捨て指定がない場合は，エラーメッセージを出力して，変換処理を途中で終了します。

桁あふれ切り捨て指定の詳細については，「6.5.3(3)　桁あふれ時の処理」を参照してください。

(e) 不要文字削除指定

入力文字列データ要素の埋め字をデータとして見なすか（OFF 指定），データから除外するか（ON 指定）を指定します。フォーマットが XML の場合は，指定できません。入力 XML の不要文字の扱いは，実行時オプションの -XWS で指定してください。

出力データ要素に対しては，不要な文字列の後続スペース，数値の前 0，小数部の下位桁の 0（有効数字以外の 0）を削除して出力するか（ON 指定），削除しないでそのまま出力するか（OFF 指定）の指定になります。

日付時刻型に対して不要文字の削除を指定する場合は，型全体に対する指定と秒の部分に対する指定を分けて定義できます。

(f) 文字コード指定

フォーマット全体で適用する文字コードを指定します。ここで指定した文字コードが，フォーマットの初期値の文字コードになります。文字コードの種類，適用範囲，及び初期値について次の表に示します。

表 6-2 文字コード一覧

文字コード	適用範囲	初期値
1 バイト文字コード	1 バイト文字列	JIS8
2 バイト文字コード	2 バイト文字列	JISK
混在文字コード	混在文字列 日付時刻型※ 1	SJIS
数値文字コード※ 2	文字列型数値属性	JIS8
セパレータ文字コード※ 2	セパレータ文字	JIS8

注※ 1

2 バイト文字コードは使用できません。

注※ 2

Interschema バージョン 1 で作成されたフォーマットに対しては，1 バイト文字コードと同じ文字コードが設定されます。

入力データ中に指定された文字コードで許されていない文字があった場合，トランスレータはエラーにするか，不正文字部分を対応する文字コードのスペースに置き換え，ワーニングエラーを出力します。ただし，EDI 標準に従って変換する場合，ヘッダ情報などで使用できる文字コードが規定されている場合があります。この場合，規定外の文字が指定された場合は，エラーになります。

(g) サイズ系指定

文字列型のサイズを数える方法として，バイト数又は文字数が指定できます。

バイト数を指定した場合，データ中のシフトコード，解放文字などもサイズとして数えます。文字数を指定した場合，シフトコードと解放文字は文字数として数えません。

(2) 型定義情報

トランスレータが扱うことができる要素の型は，数値（文字列型，バイト列型），文字

列，バイト列，日付時刻です。

要素の型属性の定義には，数値，文字列などの主属性の指定，右寄せ・左寄せと埋め字の指定，要素サイズの指定，文字コードの指定，及びトランスレータへの指示項目（桁あふれ時の処理など）があります。また，要素データの離散値を定義できます。

文字コード指定，不要文字削除指定，サイズ系指定については，「(1) フォーマット情報」の該当項目を参照してください。

型定義情報を次に示します。

表 6-3 型定義情報

型定義情報	定義箇所	指定する内容
右寄せ・左寄せと埋め字の指定	型	要素データが指定されたサイズに満たない場合，要素エリアのどちら側に詰めて表現するかを指定します。また，不足サイズ部分の埋め字を指定できます。※
サイズ定義	型	要素の固定サイズを指定するか，可変長にするかを指定します。
値定義	型	要素が取る値を指定します。

注※

数値属性は右寄せして埋め字：0，文字列属性は左寄せして埋め字：スペースが一般的です。

各型定義情報の詳細について説明します。

(a) 右寄せ・左寄せと埋め字の指定

要素データが指定されたサイズに満たない場合，要素エリアのどちら側に詰めて表現するかを指定します。また，不足サイズ部分の埋め字も指定できます。ただし，2 バイト文字コードの文字列の埋め字は 2 バイト文字，その他の属性に対する埋め字は 1 バイト文字を指定してください。また，文字コードが EUC 又は UTF8 の場合，埋め字に半角仮名は指定できません。

数値に対する埋め字は，0 又はスペースが指定できます。ただし，桁セパレータを使用する場合の埋め字は，スペースだけが指定できます。

埋め字は，文字（シフト JIS）又はバイト列で指定できます。

(b) サイズ定義

要素の固定サイズを指定するか，最大・最小サイズを指定して可変長にするかを指定します。要素サイズを可変長にした場合，サイズを動的に決める指定をするか，セパレータを使用して要素の終端を明確にします。

サイズを動的に決める指定は，その要素が使われている構造内で，サイズをサイズ決定式として宣言します※ 1。例えば，入力データがレングスタグ構造を持つ場合，データのサイズは，その前にあるレングスを表す要素の値となります。サイズ決定式については，

「6.6.2 サイズ決定式」を参照してください。

文字列以外の要素のサイズは，バイト数で指定します。文字列要素のサイズは，バイト数又は文字数で指定します。詳細については，「6.1.1(1)(g) サイズ系指定」を参照してください)。数値型の要素のサイズは，符号や指数記号などを桁数に含めるかどうか，主属性の定義内で指定できます※2。

注※1

出力フォーマットの場合は，要素データサイズは出力内容（結果）で決まるため，サイズ決定式は定義しなくてもかまいません。

注※2

FDL エディタでの定義では，固定長要素の場合は，符号や指数記号などを，桁数に含めます。可変長要素の場合は桁数に含めません。

(c) 値定義

要素が取る値を定義します。値定義された要素に対して，定義されていない値が入出力データ中に現れた場合，トランスレータはエラーとして処理します。入力値をそのまま出力側へ代入したい場合などは，値エラー検証の解除を指定することもできます。

また，マップ式で VALUEMAP 関数を使用して，入力要素と出力要素の値同士をマッピングして変換できます。

日付時刻型の要素に対しては，値定義できません。

(3) セパレータ定義情報

セパレータ定義情報を次に示します。

表 6-4 セパレータ定義方法

セパレータ定義方法	指定する内容
セパレータ名による指定	要素の型属性定義のように，構造内で使用するセパレータに名前を付けて指定します。※1
セパレータ値の直接指定	構造のプロパティとして，セパレータの値を直接指定します。※2

注※1

セパレータを使用する構造は，このセパレータ名称を用いてセパレータの定義ができます。

注※2

セパレータを文字で指定する場合は，初期値のセパレータ文字コードを使用します。ただし，シフトコードを含む文字は指定できません。使用できない文字を次に示します。なお，JISKS は JISK，KEISS は KEIS として扱います。

JISE：

2バイト文字，1バイト仮名文字

ISOJP，KEIS_EBCDIC_MIX，KEIS_EBCDIK_MIX，IBM_EBCDIC_MIX，IBM_EBCDIK_MIX，JEF_EBCDIC_MIX，JEF_EBCDIK_MIX：

2バイト文字

セパレータは，出現する位置によって意味が異なります。セパレータの出現位置には，次の四つの種類があります。

- 構造の前に出現
  セパレータの出現で，構造であることが認識されます。データタグなどに適用できます。
- 構造の後に出現
  セパレータの出現で，構造の終了が認識されます。
- 各要素の前に出現※
  セパレータの出現で，要素の区切りと認識されます。
- 各要素間に出現※
  セパレータの出現で，要素の区切りと認識されます。

注※

各要素の前に出現と各要素間に出現する場合は，構造の定義方法によって使い分ける必要があります。

セパレータ定義情報の例を次に示します。

```
TAG+DATA1+DATA2+DATA3'
（TAG＝構造の前に出現，　'＝構造の後に出現，＋＝各要素の前に出現）
A1，A2，A3，A4，A5
（，＝各要素間に出現）
```

また，解放文字もセパレータと同じく，名称及び固定の値を定義できます。これは，構造に対するセパレータの定義と同レベルで設定します。ただし，セパレータとは異なり，解放文字をネストして定義できません。解放文字は最上位のルート構造でだけ宣言できます。

セパレータを「+」，解放文字を「？」とした例を，次に示します。

```
ABC?+DEF+?+123
```

「ABC+DEF」と「+123」というデータを表します。

次に，個別に定義するセパレータとは別に，トランスレータが提供するセパレータについて説明します。トランスレータが提供するセパレータを次に示します。

- #Comma
- #Linefeed

「#Comma」は，1 バイト文字のコンマ「,」を表します。Windows の場合，「#Linefeed」は CR/LF（復帰・改行）を表します。ワークステーションの OS の場合，「#Linefeed」は LF（改行）を表します。それぞれの値は，セパレータの文字コードに従います。

(4) 変数定義情報

フォーマットの型コンポーネントとは別に，データを保持できます。グループ単位出力指定によってデータが破棄される前に，コンポーネントの値を保存する場合などに変数定義情報を利用します。

変数は，名前を付けて設定します。変数の初期値は数値，文字列，バイト列のどれかです。

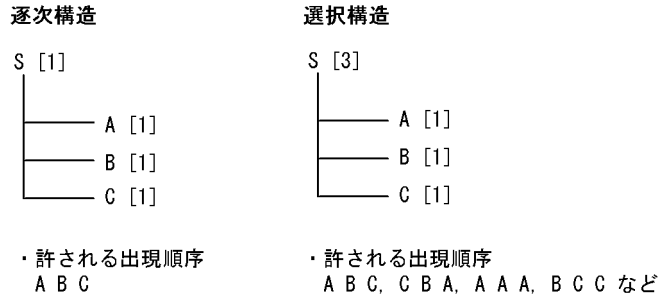
変数の値は，マップ式で参照できます。また，変数式によって変数の値を更新することもできます。変数式については，「6.6.7　変数式」を参照してください。

(5) 構造定義情報

変換対象となるデータの構造を，任意に指定できます。構造定義は，要素の出現順序と出現回数を定義するものです（C 言語の構造体に相当します）。定義する情報は，構造の名称，種類，コンポーネント（構成要素）の情報，評価規則，セパレータなどです。セパレータについては，「(3)　セパレータ定義情報」を参照してください。

構造の種類には，逐次構造と選択構造があります。逐次構造は，構成メンバが定義された順序に出現する構造，選択構造は，構成メンバのどれか一つが出現する構造を表します。構成メンバには，型属性定義されている要素か，同じフォーマット内で定義される構造を設定できます。

逐次構造と選択構造の例を次に示します。



逐次構造の場合は，ツリーのとおりに出現します。選択構造の場合は，構造 S の子コンポーネントの出現順序は不定で，特に条件式などが定義されていなければ先頭に定義されたコンポーネントが選択されます。子コンポーネントを選択するための条件は，選択構造に対して定義する評価順序決定式と，子コンポーネントに対して定義する順序決定値のセットで指定できます。順序決定値はコンポーネント情報として定義します。

構造定義情報を次に示します。

表 6-5 構造定義情報

構造定義情報	定義箇所	指定する内容
順序決定式～順序決定値	構造（親コンポーネント）	選択する条件が決まっている場合に，選択順序の変更を指定します。
条件式	構造及びコンポーネント	構造やコンポーネントに対して，その値や出現状態の条件を記述します。

各構造定義情報の詳細について説明します。

(a) 順序決定式～順序決定値

選択構造を持つコンポーネントのメンバは，FDL ファイル及び MDL ファイルに記述された順序で評価され，最初に正当であると評価されたものが選ばれます。これに対して，メンバを選択する条件が決まっている場合，選択順序を変更できます。

評価順序指定があって，指定条件が値のどれにも合致しなかった場合は，順序決定値が定義されていない子コンポーネントについて，従来のメンバの定義順に従って評価します。値が合致した選択構造の子コンポーネントとデータが不整合の場合は，エラーとして処理します。

(b) 条件式

評価規則として条件式が定義されていた場合，入出力データに対して評価した結果が偽と判定した場合，トランスレータはエラーとして処理します。

構造に対して定義した条件式は，その構造の子コンポーネントがすべて正当なときに評価されます。コンポーネントに対して定義した条件式は，該当するコンポーネント自身（構造コンポーネントならば，その子コンポーネントすべて）が正当であったときに評価されます。

(6) コンポーネント定義情報

コンポーネント情報は，型や構造などをどのように使用するかを定義するものです。型や構造の出現順序，出現回数など，データ構造の枠組みを規定する情報や，順序決定値，サイズ決定式など，コンポーネント独自の性質を決める情報などがあります。また，コンポーネントの正当性を評価するための評価規則なども，コンポーネント定義情報です。

コンポーネントは構造の子として定義します。型属性定義されている要素か，同じフォーマット内で定義される構造です。実際のデータの並びに合わせて，コンポーネントの順序を定義します。同時に，出現回数（繰り返し数）を定義する必要があります。

出現回数は最小数，最大数を指定して表します。最小数に 0 を指定すると，そのコンポーネントは入出力データ中に現れなくてもかまいません。最小数と最大数を同じにした場合，そのコンポーネントは指定された数だけ出現しなければなりません。出現数が不足する場合，トランスレータはエラーとして処理します。

コンポーネントの出現回数を動的に決定する定義もできます。選択構造の子コンポーネントの最小出現回数は，原則として1以上にしてください。

各コンポーネント定義情報の詳細について説明します。

表 6-6 コンポーネント定義情報

コンポーネント定義情報	指定する内容
必須項目の指定	コンポーネントが必須項目であることを指定します。※
カウンタの指定	コンポーネントの出現回数を取得するカウンタを指定します。
出現回数決定式	可変回出現コンポーネントの出現数を動的に決める指定をします。出現数を動的に決める指定と出現回数決定式（算術式）を組にして指定します。出力フォーマットに対しては，出現数を動的に決める指定だけでもかまいません。
サイズ決定式	可変長でサイズを動的に決定する定義の型コンポーネントのサイズを，算術式として指定します。出力フォーマットに対しては，指定しなくてもかまいません。
デフォルト式	型コンポーネントの初期値を指定します。
変数式	変数の値の変更を指定します。

注※

MDL エディタ上でマップ式定義を促すためのものです。

各構造定義情報の詳細について説明します。

(a) 必須項目の指定

必須項目が指定されたコンポーネントに対しては，マップ式（又はデフォルト式）が定義されないと MDL のエラーになります（ただし，親コンポーネントにマップ式定義などがある場合）。省略できる子孫の位置づけにあるコンポーネントは，エラーになりません。

ただし，トランスレータでの変換では，この指定は意味を持ちません。必須項目の指定をしたコンポーネントの最小出現回数が0の場合，変換実行時に該当するコンポーネントが出現しなくても，トランスレータはエラーにしません。エラーにしたい場合は，別途条件式などで記述してください。

(b) カウンタの指定

カウンタの指定は，すべての種類のコンポーネントが対象となります。初期値は0です。カウンタの指定には，次の2種類があります。

- カウンタを初期化する
- カウンタをインクリメントする

カウンタを初期化する定義情報を持つコンポーネントが出現すると，カウンタは0にな

ります。その後，カウンタをインクリメントする定義情報を持つコンポーネントが出現するごとに，カウンタが一つずつ増えていきます。カウンタの初期化及びインクリメントは，マップ式などの評価後，コンポーネントが正当に出現したと判定された時に開始されます。

カウンタは，フォーマットごとに設定されます。複数のフォーマットにわたってカウントすることはできません。カウンタの値を取り出す GETCOUNT 関数については，「6.7.3　関数」を参照してください。

(c) 出現回数決定式

トランスレータは，出現回数決定式を評価した結果の値を，コンポーネントの固定の出現数と見なします。出力フォーマットで出現回数決定式がないか，評価結果が不正である場合は，従来のコンポーネントの出現回数の定義が有効になります。

入力フォーマットに対して，EXISTWHILE 関数又は WHILE 関数を使用した出現回数決定式を記述できます。この場合，EXISTWHILE 関数又は WHILE 関数で指定した条件が満たされる間，コンポーネントが出現すると見なされます。EXISTWHILE 関数又は WHILE 関数については，「6.7.3　関数」を参照してください。

(d) サイズ決定式

トランスレータは，サイズ決定式を評価した結果の値を，型の固定のサイズと見なします。出力フォーマットでサイズ決定式がないか，評価結果が不正である場合は，型定義での最小／最大サイズの定義が有効となります。

(e) デフォルト式

通常，出力側フォーマットの 1 回以上出現するコンポーネントに対してマップ式が定義されていなければなりません。これに対して，あらかじめコンポーネントの初期値を設定できます。

初期値が設定されたコンポーネントにマップ式が定義されなかった場合，トランスレータはエラーにしないで，初期値として設定されている固定の値を出力します。

デフォルト式はマップ式と同様に，個別に指定できます。

(f) 変数式

変数式は，変数の値を更新するときに使用します。一つのコンポーネントに対して複数の変数式を定義できます。変数式は，コンポーネントが 1 回出現した時点で評価されます。指定方法はマップ式と同様で，全体指定と個別指定の 2 種類があります。

6.1.2 MDL での定義情報

MDL では，マップ式以外にもグループ単位出力と未使用コンポーネントについて定義できます。ここでは，グループ単位出力指定と未使用コンポーネント指定について説明します。

(1) グループ単位出力指定

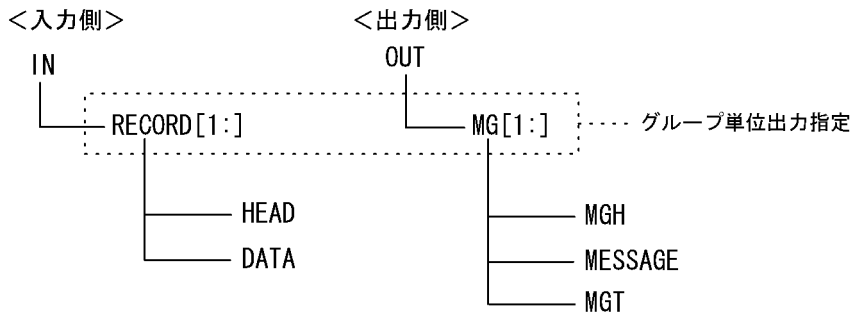
MDL エディタで指定できるグループ単位出力の指定について説明します。

通常，トランスレータは，入力データをすべて読み込んでから変換処理を実行し，最後に変換結果を出力します。この場合，1 か所でもデータに不正があると，変換全体が失敗することになります。また，入力データを一括して読み込むため，データのサイズが大きくなると，メモリが不足するおそれがあります。しかし，グループ単位での変換の実行を指定すると，変換途中でエラーが発生しても，それまでに正常に変換されたグループの出力データは保証されます。したがって，エラー原因を修正した後は，正常に出力できたグループ以降のグループから変換処理を続行できます。このような指定を「グループ単位出力指定」と呼びます。グループ単位出力指定では，CII 標準形式のメッセージグループのように，意味を持つグループ単位でデータを変換できます。さらに，1 グループ単位で入力や出力をするため，メモリ使用量を抑えられます。

グループ単位出力指定は，MDL エディタで入力フォーマットのグループを表すコンポーネントと，出力フォーマットのグループを表すコンポーネントとを対応付けて，対で指定します。以上のように指定しておくと，トランスレータは入力側の 1 グループのデータを読み込んで変換処理を実行し，出力側の 1 グループのデータを出力します。

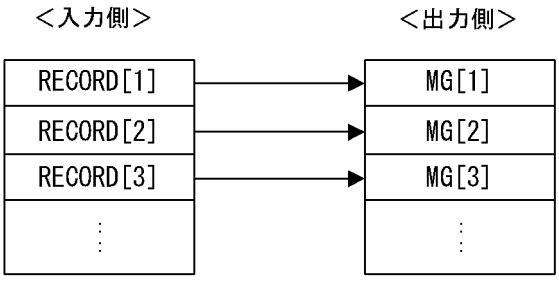
次のような構造のフォーマットがあります。

図 6-1　グループ単位出力指定の構造の例



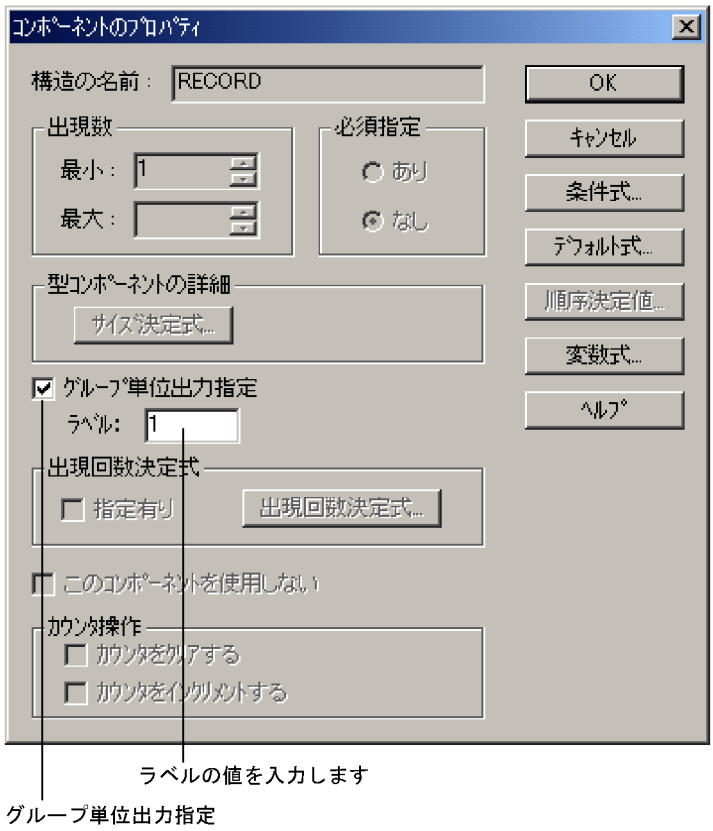
入力側のコンポーネント「RECORD」と出力側のコンポーネント「MG」に対して，グループ単位出力を指定すると，入力側のコンポーネント「RECORD」が 1 回出現するごとにデータが変換されます。「図 6-1　グループ単位出力指定の構造の例」のフォーマットのデータ変換処理を次に示します。

図 6-2　グループ単位出力指定のデータ変換の例



グループ単位出力指定を入力側のフォーマットだけに対して指定した場合は機能しません。出力側のフォーマット，又は入力フォーマットと出力のフォーマットを組にして指定してください。グループ単位出力指定を出力側のフォーマットだけに対して指定した場合は，入力データを一括して読み込み，指定されたコンポーネントごとに変換結果をファイル（又はメモリ）に出力します。

グループ単位出力指定は，［コンポーネントのプロパティ］ダイアログで「グループ単位出力指定」をチェックして指定します。



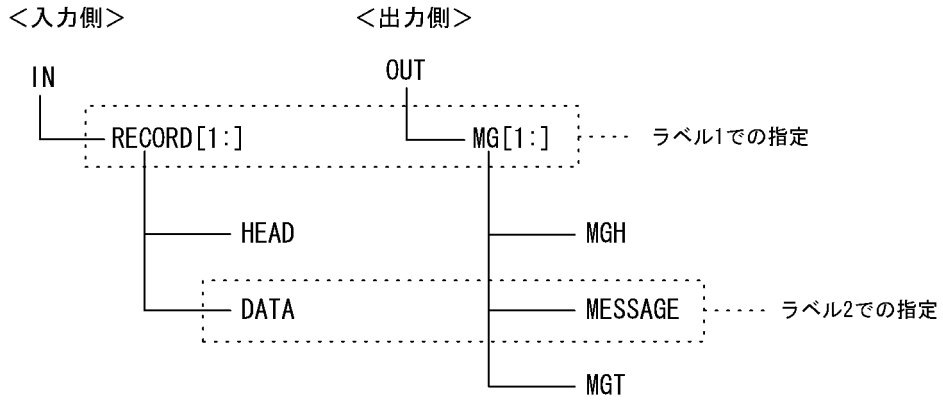
グループ単位出力指定は，ラベルを付けて複数階層に指定できます。ラベルを使用する場合は，［コンポーネントのプロパティ］ダイアログで「グループ単位出力指定」をチェックし，ラベルの値を入力します。ラベルの値は 1 から指定し，対応する入力側と出力側のコンポーネントには同じ値を入力してください。

ラベルを使用する場合，次の二つの条件を満たす必要があります。

- データを出力するグループは，最上位に指定されたコンポーネントに対応する。
- 上位のグループから下位のグループまでが包含関係にある。

ラベルを使用したグループ単位出力指定の例を次に示します。

図 6-3　グループ単位出力指定のラベルを使用した例



グループ単位出力指定は，大量のデータを一括変換する場合に有効な指定です。グループとして指定するコンポーネントは，可変回出現する最上位のコンポーネントを目安にしてください。なお，グループ単位出力を指定した出力側のコンポーネントにある要素の値が，1 回分のグループの入力ですべて決まらない場合はエラーになるので注意してください。

グループ単位出力指定には，次の制限事項があります。

- ラベル 1 のグループより上位の階層（親又は先祖のコンポーネント）に複数回出現するコンポーネントがある場合は指定できません。最上位の階層にラベル 1 を指定してください。
- グループ内で評価順序遅延指定のマップ式（%delay n）があり，グループより上位の階層にあるコンポーネントが含まれる場合は，指定できません。
- MDL 内に複数の入力フォーマット（又は出力フォーマット）がある場合でも，グループ単位出力指定を指定できる入力フォーマット（又は出力フォーマット）は一つだけです。
  グループ単位出力指定が指定されていない入力フォーマットは，最初のグループよりも前にデータがあると見なされます。また，グループ単位出力指定が指定されていない出力フォーマットは，最後のグループよりも後にデータを出力すると見なされます。

- 最初のグループより前にデータがあるコンポーネントを，グループ内のコンポーネントにマッピングして値を参照することはできますが，グループ内のコンポーネントを，そのグループより後に現れるコンポーネントにマッピングして値を参照することはできません。

（2） 未使用コンポーネント指定

MDL エディタで指定できる未使用コンポーネントについて説明します。

選択構造の不要なメンバなど MDL 内で使用しないコンポーネントがある場合は，未使用コンポーネントとして指定します。未使用コンポーネントに指定されたコンポーネントは，MDL 内に存在しないと見なされます。使用しないコンポーネントをあらかじめ指定しておくと，トランスレータの処理効率が向上します。未使用コンポーネントの指定方法については，「4.3.5　選択構造を持つフォーマットのマッピング」を参照してください。

なお，未使用コンポーネントとして指定できるのは，選択構造のメンバであるコンポーネントだけです。選択構造のメンバ以外のコンポーネントは，未使用コンポーネントとして指定できません。

6.2 属性

FDL エディタで要素の型を定義するときに指定する属性には，数値，文字列，バイト列，日付時刻があります。属性の指定方法については，FDL エディタのヘルプを参照してください。

各属性について次に説明します。

6.2.1 数値

数値型属性には，文字列型数値属性（整数，実数，及び暗黙的小数部付数値）と，バイト列型数値属性（整数，実数，及び暗黙的小数部付数値以外）があります。文字列型数値属性の符号は省略できます。

符号の出現位置として，数字列の前後どちらかを選択できます。符号と数字列の間にスペースがあってもかまいません（トランスレータが出力する場合，スペースは挿入されません）。入力データが負符号だけを使用する指定でも，＋符号が入力できます。入力データに数字が一つも出現しない場合は，0 と見なします。なお，埋め字は，0 又はスペースを使用できますが，左寄せ指定や桁セパレータを使用する場合は，スペースだけ使用できます。

どの数値属性も，トランスレータでは，10 進数として扱います。扱える 10 進数としての最大桁（整数部と小数部の桁数の合計）は，30 桁です。

それぞれの属性について説明します。

(1) 整数

「数字」「符号」「桁セパレータ」から構成される数値属性です。数値は最大 30 桁で小数部を持ちません。例を次に示します。

1234567890

-1,234,567,890

1234567890+

(2) 実数

「数字」「符号」「小数点」「桁セパレータ」「指数記号」から構成される，一般的な形式の数値属性です。数値は最大 30 桁（小数部，指数部桁数を含む），小数部は最大 29 桁，指数部は最大 2 桁です。入力データ中に数字がなく，小数点だけある場合や，指数部の指定だけがある場合はエラーになります。指数記号文字は，実数型が従う文字コードの中から指定してください。例えば，JIS7A コードの場合，小文字の「e」は使用できません。

次に例を示します。

1234567890

1,234,567,890+

-123.456

1.23E3　　(= 1230)

(3) 暗黙的小数部付数値

小数点が暗黙的に指定される数値で，「符号」「数字」から構成されます。数値は最大 30 桁（小数部を含む），小数部は最大 29 桁です。次に小数部が 4 桁の場合の例を示します。

1234567890　　(= 123456.789)

+123　　(= 0.0123)

1230000-　　(= -123)

(4) ゾーン形式 (アンパック形式) 数値

最大 16 桁（16 バイト）のゾーン形式数値です。小数点は暗黙的に示されます。符号は項目の最終バイトのゾーン部で表します。ゾーン形式数値の内容を次に示します。

表 6-7　ゾーン形式数値の内容

文字コード	正符号 +	負符号 −
JIS8	0x3	0x7
EBCDIK	0xC 又は 0xF	0xD

最終バイト以外のゾーン部は各コードの正符号で表し，数値は各バイトの下位 4 ビットで表します（0x0 ～ 0x9）。次に JIS コードの例を示します。

0x31323334　　(= 1234 小数部なしの場合) (＝ 12.34 小数部 2 桁の場合)

0x31323374　　(= -1234 小数部なしの場合)

EBCDIK コードの場合，トランスレータは正符号として 0xC を出力します。ただし，実行時オプション「-FDS」の指定がある場合，0xF を出力します。

この属性型に対しては，数値の桁数を指定するとサイズも指定されます。サイズ＝桁数になります。

(5) パック形式数値

最大 30 桁（16 バイト）のパック形式数値です。小数点は暗黙的に示されます。符号は項目データの最終バイトの下位 4 ビットで表します。正符号は 0xC 又は 0xF，負符号は 0xD で表し，数値は 4 ビットで表します（0x0 ～ 0x9）。桁数が偶数の場合，最上位バイ

トの上 4 ビットは 0 でなければなりません。なお，トランスレータは正符号として 0xC を出力します。

0x01234F　（= 1234 小数部なしの場合）（＝ 12.34 小数部 2 桁の場合）

0x01234D　（= -1234 小数部なしの場合）

この属性型に対しては，数値の桁数を指定するとサイズも指定されます。サイズ＝↑（桁数＋ 1）／ 2 ↑です。

（6） 符号付 2 進整数

1 ～ 4 バイトまでの符号付きの 2 進数です。エンディアンは，フォーマットの情報に従います。次にビッグエンディアンの場合の例を示します。

0x0100　（= 256）

0xFEFF　（= -257）

（7） 符号無 2 進整数

1 ～ 4 バイトまでの符号なしの 2 進数です。次に，2 バイトでビッグエンディアンの場合の例を示します。

0x0100　（= 256）

0xFEFF　（= 65279）

6.2.2 文字列

文字列型は，任意の埋め字文字を指定できます。この型が属する文字コードの文字として認識されない値でもかまいません。通常，埋め字はスペース文字で，データとしては扱いません。また，出力する文字列データの末尾スペース文字は，不要文字の扱いとなります。埋め字，不要文字の扱いは，「6.1.1(1)　フォーマット情報」及び「6.1.1(2)　型定義情報」を参照してください。

データのサイズは，バイト数又は文字数を指定できます。文字数には，シフトコードや解放文字を含みません。

（1） 混在文字列

1 バイト／ 2 バイト文字コード又は混在文字コードで表される文字列です。次にシフト JIS コード（SJIS）の例を示します。

123 漢字 ABC

(2) 1 バイト文字列

1 バイト文字コードで表される文字列です。次に 8 ビット JIS コード（JIS8）の例を示します。

123ｶﾝｼﾞ abc

(3) 2 バイト文字列

2 バイト文字コードで表される文字列です。次に JIS 漢字コード（JISK）の例を示します。

１２３漢字ＡＢＣ

6.2.3 バイト列

プラットフォーム間でビットパターンが保存されるバイト列です。

出力データが要素の最小サイズに満たない場合，トランスレータはデータを左寄せで出力し，余りを 0x00 で埋めます。

6.2.4 日付時刻

年，月，日，時，分，秒のパートの組み合わせで構成する型です。パート部分は数値ですが，型全体としては文字列と同じ扱いとなり，左右寄せ，埋め字が任意に指定できます。ただし，パート値として使用できるのは 1 バイト文字で，文字コードとして 2 バイト文字コードは使用できません。文字コードとして初期値が指定された場合は，フォーマットの混在型文字列の文字コードに従います。

パートについては，「(3)　日付時刻型」を参照してください。

(1) 日付

8 桁（CCYYMMDD），6 桁（YYMMDD）の数字によって，年月日を表します。MM は 01 ～ 12，DD は 01 ～ 31 の範囲です。

パート間の区切り文字がなく，パート値の前 0 ありで構成された日付時刻型の一形態です。

CCYYMMDD

最大 8 桁の数字によって，日付を表す形式です。上位桁が省略された場合は，その桁は 0 と見なします。次に例を示します。
19980520 （＝ 1998 年 5 月 20 日）

YYMMDD

最大 6 桁の数字によって，1951 年～ 2050 年の間の日付を表す形式です。上位 2 桁

「YY」が 00 ～ 50 の場合は 2000 年から 2050 年までを表し，51 ～ 99 の場合は 1951 年から 1999 年までを表します。上位桁が省略された場合は，その桁は 0 と見なします。次に例を示します。
010907　（＝ 2001 年 9 月 7 日）
980520　（＝ 1998 年 5 月 20 日）

(2) 時刻

最大 6 桁の数字によって，24 時間制の時刻（時分秒）を表します。数字は 6 桁（hhmmss）です。hh は 00 ～ 23，mm は 00 ～ 59，ss は 00 ～ 59 の範囲です。上位桁が省略された場合は，その桁は 0 と見なします。

パート間の区切り文字がなく，パート値の前 0 あり，秒小数部なしで構成された日付時刻型の一形態です。次に例を示します。

133000　（＝ 13 時 30 分 0 秒）

(3) 日付時刻型

日付時刻型は，数値を表すパート，各パート間を区切る文字，ゾーン部から構成されています。秒を表すパートは，小数部を持つことができます。パートの順序構成は任意です。ただし，ゾーン部は型の末尾です。個々のパートは数値で，前0の有無が指定できます。

「CC」を使用しないで「YY」だけを使用したとき，「YY」が 00 ～ 50 の場合は 2000 年～ 2050 年までを表し，「YY」が 51 ～ 99 の場合は 1951 年～ 1999 年までを表します。

トランスレータは，通常はパート値の検証をしません。各パートの値の妥当性は実行時オプション「-DTM」が指定された場合だけ実行します。ただし，実行時オプションの指定があっても，日時としての妥当性は検証しません。

日付型，時刻型を含め，日付時刻型には，次の制限事項があります。

- パート間の区切り文字，ゾーン部の先頭文字として数字文字（小数点文字を含む）は使用できません。
- 小数点として認識する文字は「.」と「,」だけです。
- 「CCYY」と，「CC」又は「YY」は同時に使用できません。
- 型の文字コードとして，2 バイト文字コードは使用できません。
- 値定義はできません。

日付時刻型のパートを次に示します。

表 6-8　日付時刻型のパート

パート	意味	備考
CCYY	西暦年（0 ～ 9999）※	1 ～ 4 桁
CC	西暦年上位 2 桁（0 ～ 99）※	最大 2 桁

パート	意味	備考
YY	年（西暦年下 2 桁）（0 ～ 99）※	最大 2 桁
MM	月（1 ～ 12）	最大 2 桁
DD	日（1 ～ 31）	最大 2 桁
hh	時（0 ～ 23）	最大 2 桁
mm	分（0 ～ 59）	最大 2 桁
ss	秒（0 ～ 59，小数部可）	整数部最大 2 桁，小数部最大 28 桁

注※

日付型の「YYMMDD」形式の場合は，西暦年換算で 1951 年～ 2050 年になります。

次に例を示します。

```
20010124     （＝2001年1月24日）
'01/01/24    （＝2001年1月24日）
2001-01-24T13:20:00+09:30※（＝2001年1月24日13時20分00秒）
```

注※

「+09:30」がゾーン部文字列です。

6.3 文字コード

トランスレータで扱う文字コードと，文字コード使用時の注意事項について説明します。変換するデータの文字コードは，フォーマットに対して指定する（デフォルト）か，各要素の型属性定義内で個別に指定します。

文字コードの指定方法については，FDL エディタのヘルプを参照してください。

6.3.1 トランスレータが扱う文字コード

トランスレータが扱う文字コードには，次の種別があります。

- 1 バイト文字コード
- 2 バイト文字コード
- 1 バイト／2 バイト混在文字コード

次にこれらの文字コードについて説明します。なお，文字コードの詳細な説明については，「付録 K.1　文字コード一覧」を参照してください。

表 6-9　文字コード一覧

コード指定子	文字コード	種別	備考
EBCDIC	日立 EBCDIC コード	1 バイト	－
EBCDIK	日立 EBCDIK コード	1 バイト	－
IBM_EBCDIC	IBM EBCDIC コード	1 バイト	－
IBM_EBCDIK	IBM EBCDIK コード	1 バイト	－
ISO8859	ISO 8859 対応コード	1 バイト	0x00 ～ 0x7F は JIS7 として扱われます。0xA0 ～ 0xFF の値はほかの文字コードに変換できません。
JIS8	8bit JIS X 0201(ISO646)，カナあり	1 バイト	－
JIS7	7bit JIS X0201(ISO646)，カナなし	1 バイト	JIS8 のサブセット
JIS7A	7bit JIS X0201(ISO646)，英小文字と制御コードを除く	1 バイト	JIS7 のサブセット
CJIS	CII 用漢字コード（JIS X 0208 + JIS X 0212 の組み合わせ）	2 バイト	CII 2.1 以降
IBM	IBM 漢字コード	2 バイト	IBM83
JEF	富士通漢字コード	2 バイト	－
JISK	JIS X 0208	2 バイト	－

コード指定子	文字コード	種別	備考
JISKS	JIS X 0208 開始コード , 終了コード付き 開始コード：0x1B2442, 終了コード：0x1B284A	2 バイト（後続 1 バイトスペースだけ指定できます）	―
KEIS	KEIS コード	2 バイト	KEIS83 (0x4040, 0xA1A1 はスペース扱い)
KEISS	KEIS コード 開始コード , 終了コード付き 開始コード：0x0A42 終了コード：0x0A41	2 バイト（後続 1 バイトスペースだけ指定できます）	KEIS83 (0x4040, 0xA1A1 はスペース扱い)
SJISK	シフト JIS コード 2 バイトだけ	2 バイト	―
EUC	Extended UNIX コード	混在	日本語
IBM_EBCDIC_MIX	IBM+IBM_EBCDIC 混在 シフト IN,OUT 付き シフト IN：0x0F シフト OUT：0x0E	混在	―
IBM_EBCDIK_MIX	IBM+IBM_EBCDIK 混在 シフト IN,OUT 付き シフト IN：0x0F シフト OUT：0x0E	混在	―
ISOJP	ISO2022-JP(拡張手法 JIS X 0202 による ASCII+JIS X0208 の組み合わせ)	混在	―
JEF_EBCDIC_MIX	JEF+IBM_EBCDIC 混在 シフト IN,OUT 付き シフト IN：0x29 シフト OUT：0x28	混在	―
JEF_EBCDIK_MIX	JEF+IBM_EBCDIK 混在 シフト IN,OUT 付き シフト IN：0x29 シフト OUT：0x28	混在	―
JISE	拡張手法 JIS X 0202 による JIS X0201+JIS X0208+JIS X0212 の組み合わせ	混在	JEDICOS 対応
KEIS_EBCDIC_MIX	KEIS+EBCDIC 混在 シフト IN,OUT 付き シフト IN：0x0A41 シフト OUT：0x0A42	混在	―
KEIS_EBCDIK_MIX	KEIS+EBCDIK 混在 シフト IN,OUT 付き シフト IN：0x0A41 シフト OUT：0x0A42	混在	―
UCS4	UniCode (UCS-4)	混在	―
UNIC	UniCode (UCS-2)	混在	―
UTF16	UniCode (UTF-16)	混在	―
UTF8	UniCode (UTF-8)	混在	―
SJIS	シフト JIS コード	混在	―

（凡例）

1バイト：1バイト文字コードです。
2バイト：2バイト文字コードです。
混在：1バイト／2バイト混在文字コードです。
－：該当しません。

要素が混在文字列型の場合は，表 6-9 のすべてのコードを使用できます。1バイト文字列，2バイト文字列の場合は，対応するコードだけ指定できます。文字列型数値属性の場合は，表 6-9 の 1 バイト文字列コード，UNIC，UTF16，又は UCS4 を指定できます。セパレータに対する文字コードは，表 6-9 のすべてのコードを指定できます。ただし，シフトコードを伴う文字をセパレータとして定義することはできません。

6.3.2 文字コード使用時の注意点

各文字コード間では，基本的に同じ種別間だけで変換できます。次に文字コード間での変換可否を示します。

表 6-10 文字コード間での変換可否

入力側／出力側	1 バイト文字コード	2 バイト文字コード	1 バイト／2 バイト混在文字コード
1 バイト文字コード	○	×	○
2 バイト文字コード	×	○	○
1 バイト／2 バイト混在文字コード	△	△	○

（凡例）

○：変換できます。
×：変換できません。
△：条件によって変換できます。

1 バイト／2 バイト混在文字コード型から 1 バイト文字コード型への変換は，文字列の内容がすべて 1 バイト文字の場合だけ変換できます。1 バイト／2 バイト混在文字コード型から 2 バイト文字コード型への変換についても同様です。

対象文字列中にユーザ定義文字（外字）がある場合で，コード変換が発生しないとき（入出力とも同じ文字コードのとき）は，そのままの値を受け付けます。文字コード変換が発生する場合は，不正文字の扱いとなります。なお，入出力が XML の場合，トランスレータ内部では UTF-16 の扱いとなります。例えば，XML 文書とローカルフォーマットが共に SJIS である場合も，トランスレータ内部では文字コード変換が発生します。そのため，入出力が XML の場合は注意してください。

また，EDI 標準規格などと対応させて文字コードを使用する場合の注意事項について説明します。

(1) XML

XML 文書として使用できる文字コードを次に示します。

- シフト JIS
- UTF-8
- UTF-16
- ISO-10646-UCS-4
- ISO-10646-UCS-2
- EUC-JP
- ISO-2022-JP
- US-ASCII

XML データを変換する場合，XML 文書のエンコーディングに関係なく，UTF-16 へ変換してデータ変換されます。

(2) CII

CII のバージョンによって，使用する文字コードが異なります。トランスレータは，CII1.51，CII2.10，及び CII3.00 に対応しています。

(a) CII1.51

原則として JIS8，JISK を使用します。メッセージグループ・ヘッダ（MGH），メッセージグループ・トレーラ（MGT）は「JIS7A」を使用します。

CII データの送受信者間でローカルな協定が結ばれている場合だけ，その他の 1 バイト，2 バイト文字コードが使用できます。

(b) CII2.10

原則として，X 属性に対しては JIS8 を，K 属性に対しては CJIS を使用します。オプションとして，X 属性に対して SJIS を，K 属性に対して UNIC を使用できます。

CII データの送受信者間でローカルな協定が結ばれている場合だけ，その他の 1 バイト，2 バイト文字コードが使用できます。

(c) CII3.00

原則として，X 属性に対しては JIS8 を，K 属性に対しては CJIS を使用します。オプションとして，X 属性に対して SJIS を，K 属性に対して SJISK 又は UNIC を使用できます。

CII データの送受信者間でローカルな協定が結ばれている場合だけ，その他の 1 バイト，2 バイト文字コードが使用できます。

（3） UN/EDIFACT

トランスレータが対応する UN/EDIFACT の文字セットと，文字セットに対応する文字コードは次のとおりです。

- 文字セット「UNOA」を使用する場合は「JIS7A」
- 文字セット「UNOB」を使用する場合は「JIS7」
- 文字セット「UNOX」を使用する場合は「ISOJP」

上記以外のラテン文字，日本語以外の 2 バイト文字コードなどの UN/EDIFACT の文字セットには，トランスレータは対応しません。

（4） Sea-NACCS

日本語（2 バイト文字）を使用する項目を除き，Sea-NACCS EDI 電文，Sea-NACCS EDIFACT 電文ともに「JIS7A」を使用します。日本語（2 バイト文字）を使用する項目は「EUC」を使用します。

（5） JEDICOS

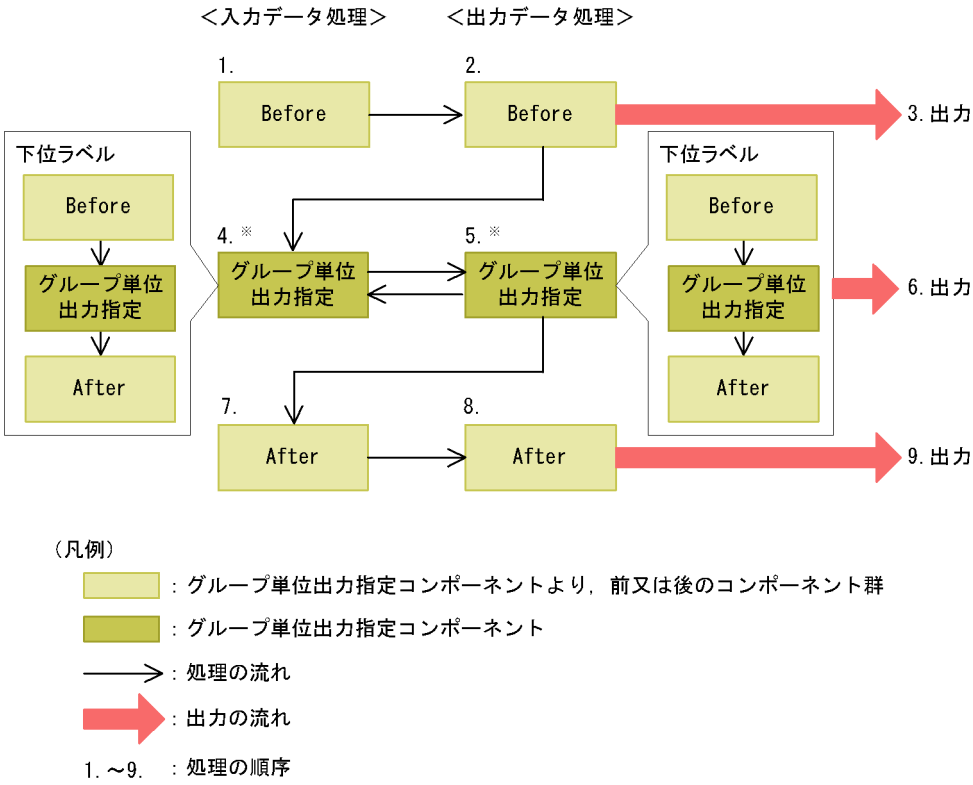
必ず「JISE」を使用します。EDIFACT の文字セットは UNOX になります。

6.4　変換処理の流れ

通常トランスレータは，入力データをすべてメモリへ展開した後，出力データを作成して，ファイルやメモリに出力します。グループ単位出力指定をした場合は，ラベルが指定されたコンポーネントの繰り返しと，その前後のコンポーネントに分けて，入力データのメモリ展開及び出力処理が行われます。

グループ単位出力指定をした場合の変換処理の流れを図 6-4 に示します。

図 6-4　グループ単位出力指定をした場合の変換処理の流れ



注※　1回出現するごとにデータが変換され，処理が繰り返されます。

図 6-4 の「Before」は，グループ単位出力指定コンポーネントより前に位置するコンポーネント群です。

入力フォーマットが複数ある場合は，グループ単位出力指定をしていないフォーマットも「Before」に含まれます。「After」は，グループ単位出力指定コンポーネントより後に位置するコンポーネント群です。出力フォーマットが複数ある場合は，グループ単位出力指定をしていないフォーマットも「After」に含まれます。

グループ単位出力指定コンポーネントは，1 回出現するごとにデータが変換され，処理が繰り返されます。

グループ単位出力指定を複数のラベルに設定した場合も，同様に処理されます。上位のラベルで 1 回に処理されるデータに対して，下位のラベルでは，Before コンポーネント，グループ単位出力指定コンポーネント，After コンポーネントの順に変換処理されます。

図 6-4 の 3.，6.，9. は，出力ファイルやメモリデータへの実際の出力を示しています。なお，下位ラベルのグループ単位出力指定コンポーネントの場合は，ワークファイルに出力され，最上位のグループのデータが出力される時点で，併せて出力先のファイルやメモリに出力されます。

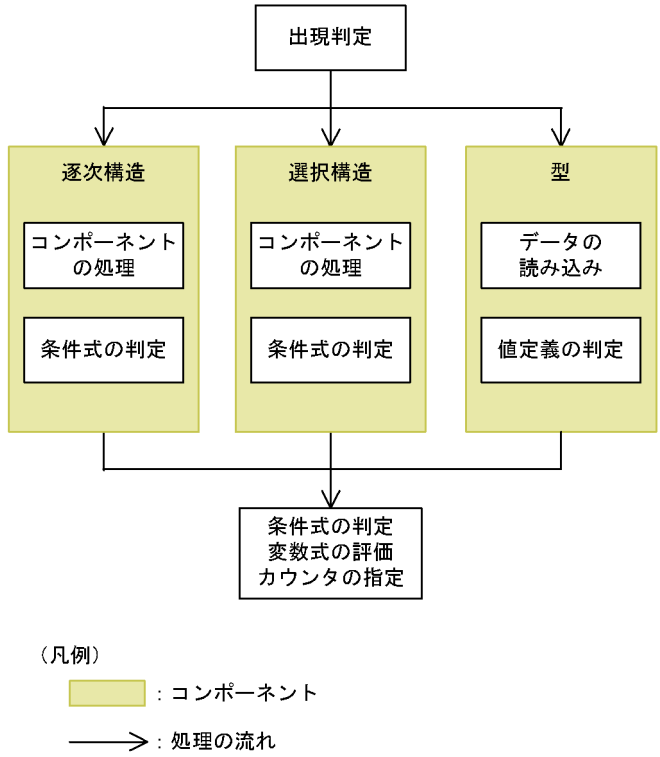
入力データのメモリ展開及び出力データの作成では，トランスレータが各コンポーネントのデータ解析や個々の型変換をして，ツリーテーブルを作成します。ツリーテーブルは，MDL に定義されたツリーの順序でコンポーネント単位に処理されます。

6.4.1 入力コンポーネントの制御

コンポーネントの出現数によって，コンポーネントの出現判定及びコンポーネントデータの作成が繰り返されます。通常は，指定した出現回数又は出現回数決定式で定義した出現回数が，コンポーネントの出現数となります。可変回出現を定義した場合は，区切りとなるセパレータが現れるまで繰り返されます。EXISTWHILE 関数，WHILE 関数などの特殊ケースで出現回数を指定した場合は，条件が満たされる間，繰り返されます。

入力コンポーネントの制御の流れを図 6-5 に示します。

図 6-5　入力コンポーネントの制御の流れ



まず，出現判定でコンポーネントが 1 回出現したと判定された場合，構造のコンポーネント及び型のコンポーネントが解析されます。

逐次構造のコンポーネントでは，構造に定義した子の順番に従って，子コンポーネントが処理されます。

選択構造のコンポーネントでは，順序決定式を定義した場合，対応する順序決定値を持つ子コンポーネントが処理されます。順序決定式を定義していない場合，子コンポーネントが順番に評価され，最初に処理対象に該当した子コンポーネントが処理されます。

構造の子コンポーネントの処理が完了した時点で，構造の条件式が判定されます。

型コンポーネントでは，型のサイズ分のデータを読み込んで，「6.7.1　データ型」で示すデータ型へ変換されます。可変長のデータの場合は，該当するセパレータが現れるまでのデータ長が型のサイズとなります。サイズ決定式を定義した場合は，サイズ決定式が評価された結果の値が型サイズとなります。データを読み込んだ後，値をチェックする場合は，値定義が判定されます。

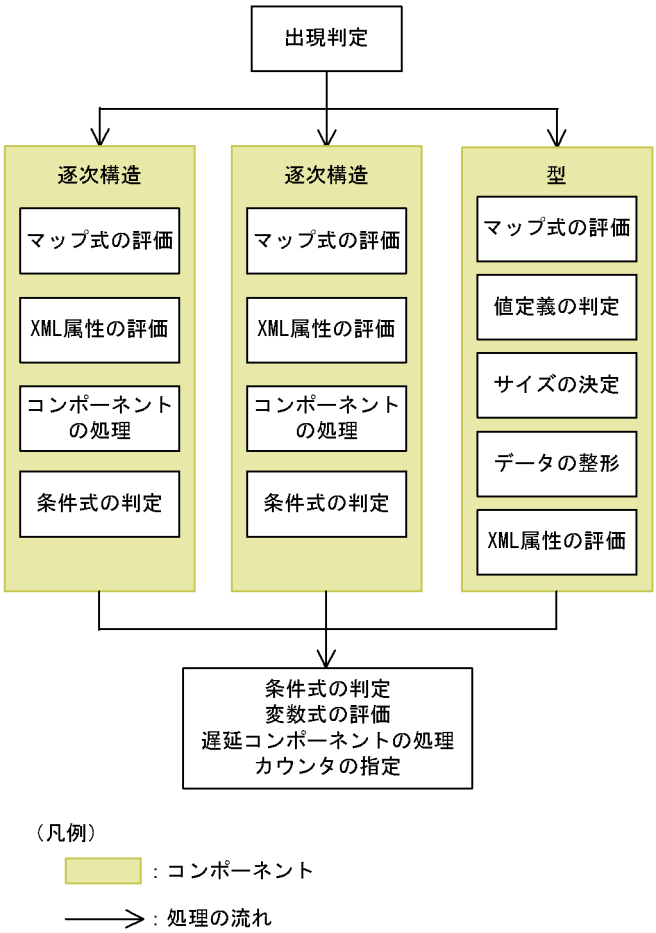
構造のコンポーネント及び型のコンポーネントの処理が完了したら，コンポーネントに定義した条件式の判定，変数式の評価，及びカウンタの指定が開始されます。

6.4.2　出力コンポーネントの制御

コンポーネントの出現数によって，コンポーネントデータの作成が繰り返されます。固定回出現のコンポーネントを除いて，通常は，マップ式の評価結果で出現の有無が判定されます。マップ式の評価結果が %UNDEF となった時点で終了となります。マップ式を定義していない場合は，最大出現数まで繰り返されます。出現回数決定式を定義した場合は，出現回数決定式が評価された結果の値が出現数となります。

出力コンポーネントの制御の流れを図 6-6 に示します。

図 6-6　出力コンポーネントの制御の流れ



まず，出現判定でコンポーネントが 1 回出現したと判定された場合，構造のコンポーネント及び型のコンポーネントが作成されます。

逐次構造のコンポーネントでは，コンポーネントに定義したマップ式，及び構造下の XML 属性を評価します。マップ式の評価結果が %UNDEF の場合，XML 属性の評価以降は処理されません。評価した後，構造に定義した子の順番に従って，子コンポーネン

トが評価及び作成されます。評価遅延（%delay）を指定した場合，子コンポーネントは評価されません。%delay %next を指定した場合は，次の兄弟コンポーネントが処理された後に評価されます。%delay n を指定した場合は，ここでは評価されないで，n が指すコンポーネントの評価後に評価されます。

選択構造のコンポーネントでは，コンポーネントに定義したマップ式，及び構造下の XML 属性を評価します。順序決定式を定義した場合は，評価された後，対応する順序決定値を持つ子コンポーネントが処理されます。順序決定式を定義していない場合，子コンポーネントが順番に評価され，最初に処理対象に該当した子コンポーネントが処理されます。

構造の子コンポーネントの処理が完了した時点で，構造の条件式が判定されます。

型コンポーネントでは，コンポーネントに定義したマップ式が評価され，コンポーネントのデータ型（「6.7.1　データ型」で示すデータ型）が求められます。値をチェックする場合は，値定義が判定されます。サイズ決定式を定義した場合は，サイズ決定式が評価された結果の値が型のサイズとなります。その後，出力データの形式に整形されます。出力データの整形後，型の下の XML 属性が評価されます。

構造のコンポーネント及び型のコンポーネントの処理が完了したら，コンポーネントに定義した条件式の判定，変数式の評価，%delay n の指定で遅延されているコンポーネントの処理，及びカウンタの指定が処理されます。

6.5 変換の規則

トランスレータが要素の値を入力や出力する場合，実際のデータと MDL で定義されたサイズに過不足があったり，ある数値型の要素の値を異なる数値型の要素へ代入したりする場合など，扱うデータ型によってトランスレータの処理が異なります。この節では，個々の場合でのトランスレータの対応について説明します。

6.5.1 異なる属性型同士での変換

基本的には，数値，文字列，バイト列，日付時刻の同じ分類の属性の型同士で変換できます。異なる分類の型に変換する場合の規則と注意事項を次に示します。

表 6-11 異なる分類の属性の型同士での変換可否

入力	出力	変換規則
数値	文字列	± n.m 形式で変換（有効数字だけ。正符号は付けない）します。結果は１バイト文字コードとなります。
	バイト列	整数部だけ 2 進整数化します。符号は常に正符号の扱いとなります。
	日付時刻	CCYYMMDDhhmmss 形式と見なして日付時刻化，ゾーン部はありません。ただし，変換先が出力フォーマットの日付型コンポーネントの場合は，CCYYMMDD，時刻型コンポーネントの場合は，hhmmss と見なして変換します。 桁が不足する場合は，該当する桁を 0 と見なします。 桁あふれ部分は無視します。
文字列	数値	± n.m 形式又は 0x で始まる（16 進表記）場合に数値化します。スペースは 0 の扱いとなります。 整数部の桁セパレータは使用できます（ただし 16 進表記以外の場合，3 桁ごとのコンマだけを認識します）。 16 進表記の場合，英大文字 / 英小文字の使用は任意で，文字列のバイト数は奇数でもかまいません。 数値として認識できない場合はエラーとなります。
	バイト列	文字列をそのままバイト列化します。
	日付時刻	CCTT-MM-DDThh:mm:ss，CCYYMMDDThhmmss，CCYYMMDDzz の場合に，日付時刻化します（zz：ゾーン部，ss：小数部可，ss 以降の文字列はゾーン部と見なします）。 その他で，CCYY-MM-DD，CCYYMMDD ならば日付型，hh:mm:ss ならば時刻型に変換します。認識できない場合は，エラーとなります。
バイト列	数値	データエンディアン※に従って 2 進整数化します。最大 60 桁です。桁あふれのときはエラーとなります。
	文字列	出力側要素の文字コードとして認識できる場合に文字列化します。認識できない場合はエラーとなります。
	日付時刻	数値型に変換した上で，日付時刻型に変換します。

入力	出力	変換規則
日付時刻	数値	日付型の場合： CCYYMMDD 形式の 10 進数化（整数 8 桁）に変換します。 時刻型の場合： hhmmss 形式の 10 進数化（整数 6 桁）に変換します。 日付時刻型の場合： CCYYMMDDhhmmss 形式の 10 進数化に変換します（整数 14 桁，SS 値に応じて小数部があり，ゾーン部は無視します）。
	文字列	日付型の場合： CCYY-MM-DD 形式に変換します。 時刻型の場合： hh：mm：ss 形式に変換します。 日付時刻型の場合： CCYY-MM-DDThh：mm：ss 形式に変換します。
	バイト列	数値に変換した上で，バイト列型に変換します。
	日付時刻	2 項演算，比較演算子に対する処理の場合だけ，日付時刻形式での数値型に変換します。

注※

式中に 16 進定数で記述されたものはビッグエンディアンの扱いになります。その他は，データが属するフォーマットのエンディアンになります。

次に，変換時に発生するエラーについて説明します。

- 変換途中でエラーが発生した場合
  データ型を変換するのは，評価する式の各項へ値を当てはめる時点，及び出力するコンポーネントの属性型に当てはめる時点です。そのため，データ型を変換した時点では正常でも，最終的に出力要素型へ変換する段階でエラーになることがあります。例えば，数値から文字列へ変換する時点では正常でも，出力先要素が 2 バイト文字コードであった場合，最終的に文字列変換ができずにエラーになります。
- 変換対象の数値の有効桁数が異なる場合
  数値型同士の変換で，入力（マップ式を評価した結果の属性型）と出力（出力コンポーネントの属性型）の有効桁数が異なるときの処理については，「6.5.3　出力データに対する規則」を参照してください。例えば，小数部を持つ数値を整数型要素へ出力する場合，あふれ部分の切り捨てを指定していなければエラーになります。なお，数値演算中の桁数は，トランスレータの最大範囲で保たれます。

6.5.2 入力データに対する規則

入力データに対する規則について説明します。

トランスレータは MDL ファイルで宣言された通りに入力データを解析し，指定された属性型の値と認識します。

入力データ中である要素として認識されるのは，MDL ファイル内のサイズ指定でされた

範囲内に記述されている内容です。例えば，入力データが「0123456789...」であるとき，先頭の要素が 3 バイト整数であれば，この要素は「012」＝ 12 となります。数値型属性の場合は，MDL ファイル内でその数値が取ることができる最大桁数が指定されています。省略時の最大桁数は，30 桁です。入力データを解析した結果，数値が最大桁数を超える場合はエラーになります。

(1) 文字列型属性の埋め字

文字列型に対して，不要文字削除指定がある場合は，埋め字部分はこの要素の値として認識しません。要素が占めるエリア内で，右寄せ／左寄せ指定がされたものとは反対側から，埋め字以外の文字が出現するまでを埋め字として認識します。次に例を示します。

入力データ　　△△漢字△漢字△△
左寄せ指定時の値　　△△漢字△漢字
右寄せ指定時の値　　漢字△漢字△△

（凡例）
△：埋め字のスペースです。

開始／終了シフトコードを使用する文字列型（JISKS，KEISS）の場合，指定された埋め字は開始から終了シフトコード内で使用されます。例外的にシフトコード外で使用できる埋め字については「(5) 開始／終了シフトコードを使用する文字列型」を参照してください。上記の例では，入力データが次のようになります。

(S)△△漢字△漢字△△(E)

（凡例）
(S)：開始シフトコードです。
(E)：終了シフトコードです。

なお，2 バイト文字列の場合は，埋め字も 2 バイトコードにしてください。

(2) 文字列型数値属性の埋め字

文字列型数値属性の場合，埋め字部分は，0 又はスペースだけ使用できます。埋め字部分は，数値の有効桁として認識されません。ただし，左寄せの場合の埋め字はスペースだけです。数値型属性の場合は，右寄せにすることをお勧めします。

例として「123.456」という値を暗黙的小数部付数値型（8 バイト，小数部 3 桁）で表すと次のようになります。

左寄せ・埋め字△　　123456△△
右寄せ・埋め字0　　00123456

数字列が省略され埋め字だけとなった場合，トランスレータは値を 0 と認識します。

(3) 日付時刻属性の埋め字

文字列型と同じです。パート部分で使用する文字を埋め字とすると，不正な結果となることがあるので，注意してください。

(4) 文字列型数値属性の入力形式

埋め字の指定にかかわらず，符号の外側（要素エリアの空き部分），符号と数字列の間にスペース文字があってもかまいません（トランスレータはスペース文字を無視するか，0として認識します）。数字列の途中でスペースなどの数字以外の文字が出現した場合はエラーになります。ただし，整数型，実数型で桁セパレータを使う指定がある場合，桁セパレータ文字は除きます。桁セパレータは整数部だけで，3桁ごとに出現しなければなりません。また，数字列の先頭が桁セパレータであってはなりません。

(5) 開始／終了シフトコードを使用する文字列型

文字列で開始・終了シフトコードを使用する文字コード型データの場合，シフトコードのほかに，1バイトのスペース文字があってもかまいません。該当する文字コードは，JISKS と KEISS です。左寄せの例を次に示します。

(S)漢字△△漢字△△(E)▼▼

（凡例）

(S)：開始シフトコードです。
(E)：終了シフトコードです。
△：埋め字として指定された文字（2バイト文字）です。
▼：1バイトのスペース文字です。

- KEIS シフト付（KEISS）の場合：EBCDIK スペース文字（0x40）
- JIS シフト付（JISKS）の場合：JIS スペース文字（0x20）

(6) 日付時刻型の規則

最終パートの後続文字列の次（後続文字列がない場合は最終パートの数値文字の次）以降を，ゾーン部文字列と見なします。数値の桁数，パート間の区切り文字，ゾーン部について，型定義とのマッチングします。秒 ss パートの小数部については，指定桁数以内で連続する数字列を小数部と見なします。各パート値について，通常は範囲を検証しません。実行時オプション「-DTM」の指定がある場合，「6.2.4(3)　日付時刻型」の「表 6-8　日付時刻型のパート」に示す範囲内であるかを検証します。省略されたパートは，値0を仮定します。

(7) その他の属性の規則

ゾーン形式数値，パック形式数値の場合，符号として使用する値（4ビット数値）が決まっています。使用できない数値が入力データ中に現れた場合はエラーになります。符号の値については「6.2.1　数値」を参照してください。

2 進整数の場合，トランスレータはフォーマットで宣言されているエンディアンに基づいて値を解釈します。初期値は，ビッグエンディアンです。

バイト列に対しては，トランスレータはサイズだけを意識し，ビットパターンは解析しません。

6.5.3 出力データに対する規則

MDL（FDL）で出力先のコンポーネントに対してサイズあふれ時の対応や，不要文字の削除が指定できます。通常，入力データを基に出力データのマップ式を評価した結果の値が，MDL で定義されている出力側の要素の型と不一致，又はサイズが超過する場合，トランスレータでエラーになります。

(1) 通常の出力形式

トランスレータは，不足サイズ分を出力側要素の型定義で指定されている埋め字で補います。

基本的にトランスレータは，文字列型数値属性で埋め字が 0 の場合は，前符号と数字列の間に埋め字を置きます。その他の属性は，出力エリアの端からデータ部分までの間に埋め字を置きます。数値は有効桁部分を出力します。ただし，最大桁数指定がある場合は，数字列部分に前 0 を付け，桁数をそろえて出力します。

最大桁数指定ありで最大桁数に満たない数値を有効桁以降だけを出力したい場合には，不要文字削除指定が必要です。

小数点は小数部桁数に含めません。

不要文字削除指定なしで，入力データ中に埋め字がある文字列を出力する場合，入力データの埋め字部分も文字列として出力します。

(2) 不要文字削除指定時の処理

不要文字削除指定をした場合，データ中から不要な文字を削除して出力します。また，数値の場合は，桁数合わせのための 0 を付けないで出力します。不要文字削除指定が有効になる属性は，文字列型数値，文字列型，及び日付時刻型です。

不要文字として削除されるのは，最小出力サイズを超える部分のスペース文字（数値型文字列，文字列，日付時刻型），又は 0（数値型文字列，日付時刻型の秒パート小数部）です。日付，時刻を文字列型数値と見なした場合の上位桁の 0 も不要文字と見なします。

暗黙的小数部付数値型の場合，この指定があっても小数部の下位桁の 0 は必ず出力します。

(3) 桁あふれ時の処理

出力するデータが指定されているサイズより大きくてもエラーにしない場合や，小数部があるデータを整数型要素に出力する場合などに指定します。原則的に，数値は小数部から切り捨て，次に上位の桁からカットします。暗黙的小数部付数値の小数部桁は残し

ます。文字列やバイト列は，データの末尾から切り捨てます。日付時刻は，文字列に組み立てた状態のデータを末尾から切り捨てます。

データの切り捨てをした場合，トランスレータはワーニングを出力します。

（4） 日付時刻型出力時の規則

ゾーン部の指定（文字列）があれば，指定されたゾーン部文字列を出力します。ゾーン部出力指定があり，ゾーン文字列の定義がない場合，入力値がゾーン部を持つ日付時刻型であれば，入力値のゾーン部文字列を出力します。その他はゾーン部なしで出力します。

各パート値については，通常は範囲を検証しません。実行時オプション「-DTM」の指定がある場合，「6.2.4(3)　日付時刻型」の「表 6-8　日付時刻型のパート」に示す範囲内であるかを検証します。

（5） その他出力時の規則

埋め字，不要文字削除，桁あふれ切り捨ての処理の優先順序を次に示します。

不要文字削除＞桁あふれ切り捨て＞埋め字

6.6　式

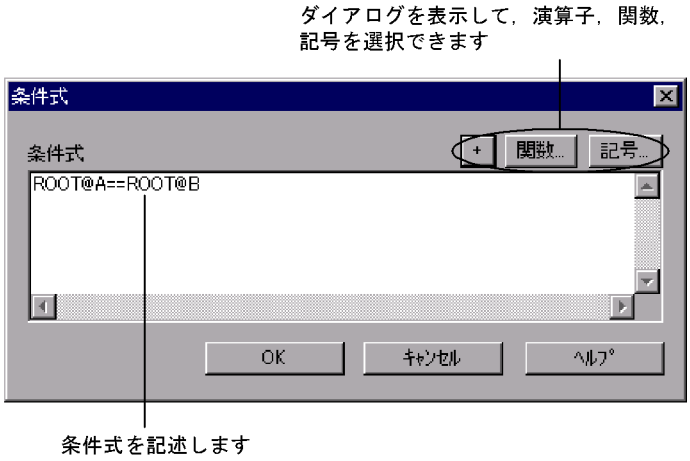
FDL エディタ及び MDL エディタで定義する式及び式中で使用できる定数とコンポーネント名について説明します。式中で使用できるのは，定数，コンポーネント名のほかに，演算子，関数などがあります。式中で使用できる演算子及び関数については「6.7　演算子及び関数」を参照してください。

FDL エディタ及び MDL エディタで定義する式のうち，条件式，サイズ決定式，順序決定式，出現回数決定式，デフォルト式，マップ式，及び変数式について説明します。

6.6.1　条件式

条件式は，式が定義された構造又はコンポーネントの正当性を評価するための評価規則式です。条件式は，入力又は出力側の構造又はコンポーネントに対して定義する式で，FDL エディタの［条件式］ダイアログに記述します。コンポーネントの条件式は，MDL エディタでも定義できます。［条件式］ダイアログは，式を定義する［構造のプロパティ］ダイアログ，又はコンポーネントの［コンポーネントのプロパティ］ダイアログで，［条件式］ボタンをクリックすると表示されます。

図 6-7　［条件式］ダイアログ



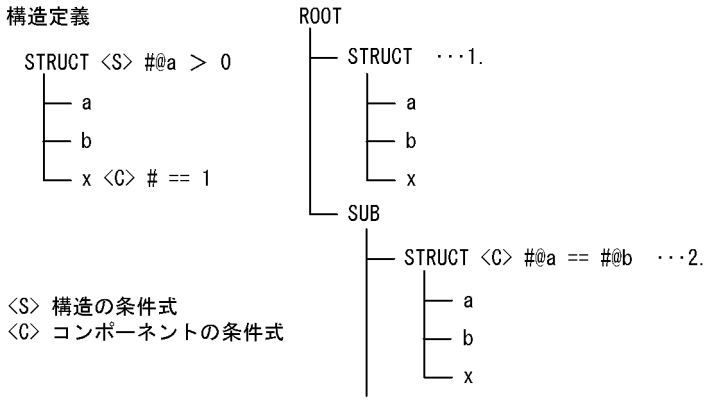
条件式は，テキストを直接入力したり，ダイアログから演算子，関数などを選択したりして記述します。

トランスレータは定義された条件式で，変換する入力データ又は出力データを評価します。条件式を評価した結果が不当な場合は，エラーになります。

構造に対して定義した条件式は，その構造がどこで使用されても必ず評価されます。一方，コンポーネントに対して定義した条件式は，そのコンポーネントに対してだけ評価されます。

次に例を示します。

図 6-8 条件式の例



構造 STRUCT 自身の構造の条件式として「#@a ＞ 0」（子コンポーネント a の値が正であること），及びその子コンポーネント x に対して，コンポーネントの条件式 「# == 1」（自身の値が 1 であること）を定義しておきます。さらに，ROOT 以下のツリーの 2. のコンポーネント部分に，構造 STRUCT に対してコンポーネントの条件式「#@a == #@b」（子の a と b の値が一致すること）を定義します。

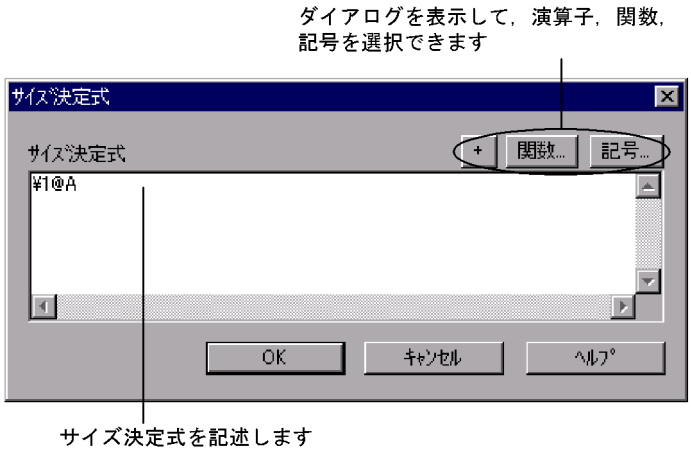
この場合，1. の構造 STRUCT のデータに対しては，「x == 1」と「a ＞ 0」の二つの条件が評価されます。一方，2. の構造 STRUCT のデータに対しては，更に「a == b」の条件を合わせて，三つの条件が評価されます。

6.6.2 サイズ決定式

サイズ決定式は，型コンポーネントのサイズが動的に決まる場合に，コンポーネントのサイズを定義する式です。サイズ決定式の評価結果によって，対応するコンポーネントのサイズが決まります。レングスタグ構造を持つデータなどに適用されます。サイズ決定式を評価した値は，正の整数でなければなりません。

サイズ決定式は通常入力側のコンポーネントに用いる式で，［サイズ決定式］ダイアログに記述します。［サイズ決定式］ダイアログは，式を定義するコンポーネントの［コンポーネントのプロパティ］ダイアログで［サイズ決定式］ボタンをクリックすると表示されます。

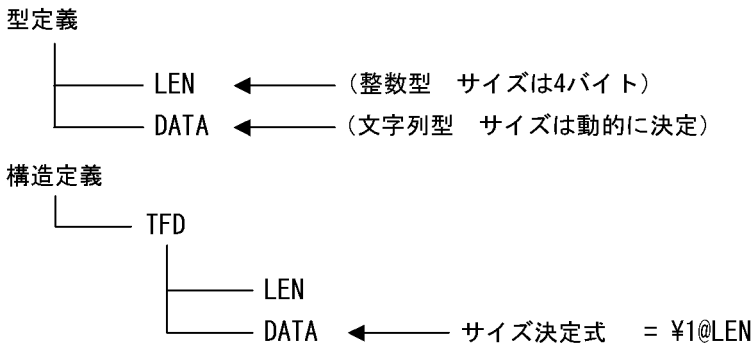
図 6-9 ［サイズ決定式］ダイアログ



サイズ決定式は，テキストを直接入力したり，ダイアログから演算子，関数などを選択したりして記述します。

次に例を示します。

図 6-10 サイズ決定式の例



入力フォーマットである場合，トランスレータはコンポーネント LEN の値をコンポーネント DATA のサイズと認識します。逆に，出力フォーマットの場合，LEN の値を DATA の固定サイズとして処理します。出力フォーマットで DATA のサイズ決定式がない場合は，コンポーネント DATA のサイズは代入される値によって決まります。

出力フォーマットでは，可変長コンポーネントのサイズは，代入元のデータのサイズに従うのが一般的です。レングスタグ構造の長さコンポーネントは，評価順序遅延指定のマップ式でデータ長を代入します。図 6-10 の例では，マップ式は次のようになります。

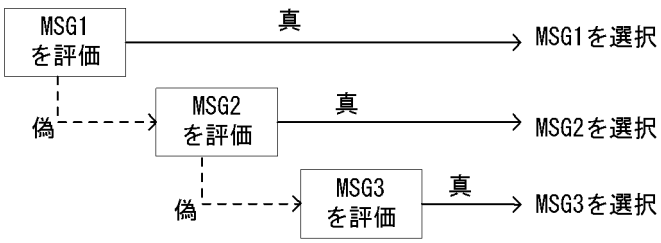
```
LENの式      %delay %next = LENGTH(¥1@DATA);
DATAの式     (任意の式);
```

6.6.3 順序決定式

選択構造の子コンポーネントは，定義された順に変換するデータとコンポーネントが合うかどうかを評価します。

例えば，選択構造「DATA」があります。この構造は，「MSG1」「MSG2」「MSG3」という子コンポーネントを持っています。

コンポーネントを MSG1，MSG2，MSG3 の順に定義した場合，まず，コンポーネント「MSG1」を評価して正当であれば「MSG1」を選択し，不当であれば「MSG2」を評価します。コンポーネント「MSG2」が正当であれば「MSG2」を選択し，不当であれば「MSG3」を評価します。つまり，選択構造のコンポーネントは定義順に評価され，最初に正当なものが選択されます。

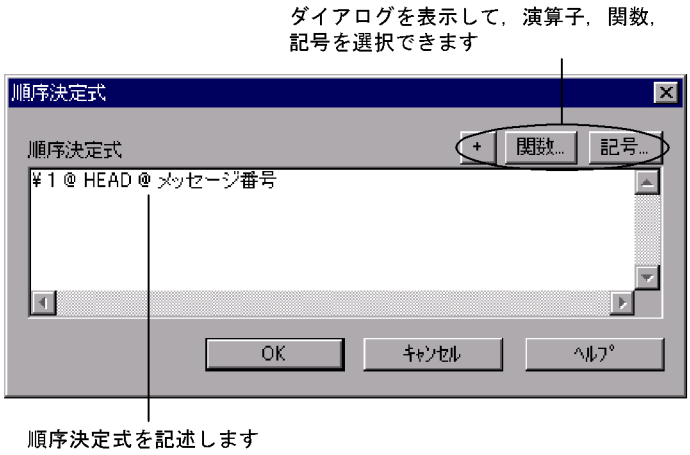


（凡例） 真：正当， 偽：不当

この評価順序は，順序決定式を定義して変更できます。データの値によって選択するコンポーネントが決まっている場合，直接そのコンポーネントを選択する指定ができます。親の選択構造のコンポーネントには評価順序の条件を表す式（順序決定式）を記述し，子コンポーネントには自コンポーネントを選択する場合の値（順序決定値）を指定します。以上のように定義しておくと，データ変換時に式を評価し，同じ値を持つコンポーネントを優先的に評価します。

評価順序を決定する式は，［順序決定式］ダイアログに記述します。［順序決定式］ダイアログは，式を定義する選択構造の［構造のプロパティ］ダイアログで［順序決定式］ボタンをクリックすると表示されます。

図 6-11 ［順序決定式］ダイアログ



順序決定式は，テキストを直接入力したり，ダイアログから演算子，関数などを選択したりして記述します。

評価順序を決定する値は，［順序決定値］ダイアログに記述します。［順序決定値］ダイアログは，値を定義するコンポーネントの［コンポーネントのプロパティ］ダイアログで［順序決定値］ボタンをクリックすると表示されます。

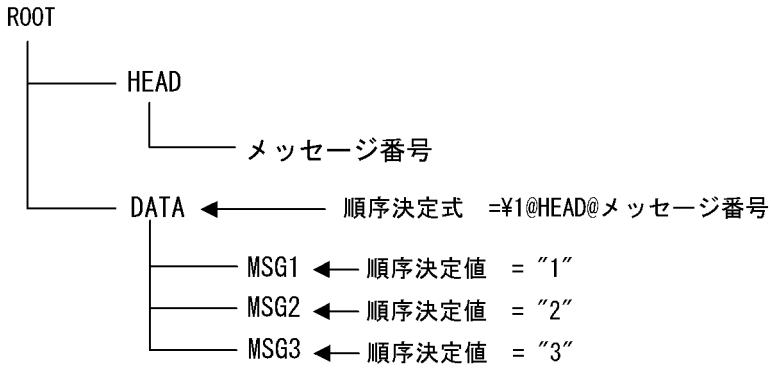
図 6-12 ［順序決定値］ダイアログ

順序決定値は，値の属性を指定して，値を直接入力します。

評価順序は対で指定するので，選択構造に対して順序決定式を定義したら，子コンポーネントに対して順序決定値を定義します（一つの選択構造に対して複数の子コンポーネントに値を定義します）。

「MSG1」「MSG2」「MSG3」というコンポーネントを持つ選択構造「DATA」の場合の例を次に示します。

図 6-13 順序決定式の例



この構造に対して，選択構造「DATA」の子コンポーネントを選択する順序決定式を定義します。

要素「メッセージ番号」の値によって子コンポーネントを選択するので，構造「DATA」に対してメッセージ番号のコンポーネント名を記述します。

```
ROOT@HEAD@メッセージ番号
```

コンポーネント名は，ルート構造から対象コンポーネントまでを「@」で連結して指定します。

また，n 階層上のコンポーネントを指定する略記法「¥n」を使用した順序決定式を次に示します。「¥1」で「DATA」の 1 階層上の構造「ROOT」を示します。

```
¥1@HEAD@メッセージ番号
```

また，子コンポーネントには順序決定式に対応する値として，メッセージ番号の値を定義します。「MSG1」に対しては「1」，「MSG2」に対しては「2」，「MSG3」に対しては「3」を定義します。以上のように定義しておくと，変換するデータのメッセージ番号が「2」の場合は，コンポーネント「MSG2」が優先的に選択されて評価されることになります。なお，選択構造の順序決定式の指定条件が，子コンポーネントの順序決定値のどれにも合わなかった場合は，次のように処理されます。

- 順序決定値が定義されていない子コンポーネントがある場合，そのコンポーネントとデータが合うかどうかを調べます。合わない場合は，エラーになります。
- 順序決定値が定義されていない子コンポーネントがない場合，選択できるものがないので，エラーになります。

次に，ORDEROFFSET 関数を使用して順序決定式を定義する方法について説明します。

通常，順序決定式では，式を定義する選択構造よりも前に出現するコンポーネントを指定します。しかし，入力フォーマットの場合，ORDEROFFSET 関数を使用して，選択条件の値にしたいコンポーネントが出現していない状態で，選択条件を指定できます。ORDEROFFSET 関数の使用例を次に示します。

図 6-14　ORDEROFFSET 関数の使用例

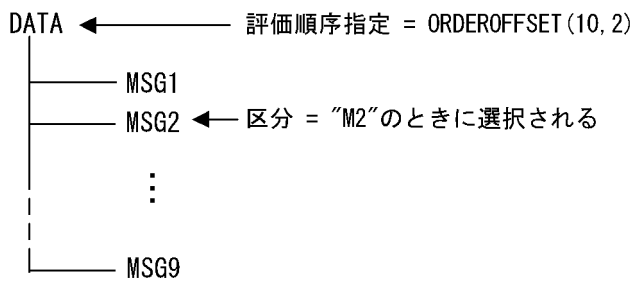
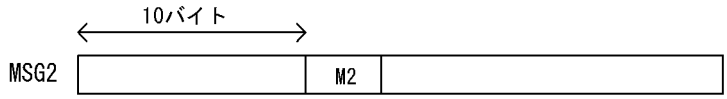


図 6-14 の例では，選択構造の子の先頭から 10 バイトの位置に，メッセージを識別するための区分があります。メッセージ MSG2 は，「M2」という区分で識別されています。



ORDEROFFSET 関数で順序決定式を指定する場合の規則を次に示します。

- 入力フォーマットで使用する。
- 順序決定式を単独の項として指定する。
- 対応する順序決定値は，バイト列として指定する。

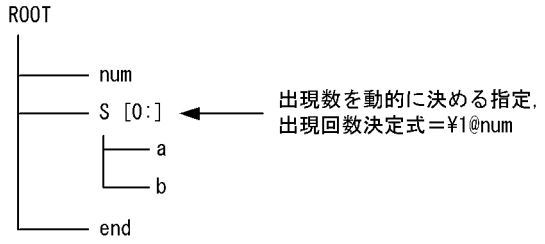
ORDEROFFSET 関数で指定した位置にデータが存在しない場合は，エラーになります。ORDEROFFSET 関数については，「6.7.3　関数」を参照してください。

6.6.4　出現回数決定式

出現回数決定式は，可変回出現コンポーネントの出現数を動的に決める場合に定義します。EXISTWHILE 関数又は WHILE 関数による式以外は，値が必ず整数値になる式を記述してください。ただし，サイズ決定式と異なり式の値は 0 でもかまいません。

型コンポーネントのサイズ決定式と同様に，入力フォーマット内では，コンポーネント出現数を動的に決める指定と，出現回数決定式を組にして定義します。出力フォーマットでは，出現数を動的に決める指定だけでもかまいません。

図 6-15 出現回数決定式の例



注 出現数を動的に決める指定がなければ，SとendのSの間に全体の終了を示すセパレータが必要です。

入力フォーマットで指定された場合，出現数を動的に決める指定がされたコンポーネントは，出現回数決定式の評価結果の値数だけデータがあるものとして処理します。上記の例では，S の出現数は num の値です。

出力フォーマットで出現数を動的に決める指定がされた場合，コンポーネントが出現回数決定式を持ち，そのコンポーネント評価時点で同式の値が決まるときは，このコンポーネントは出現回数決定式の評価値を出現数とします。出現回数決定式がないか，コンポーネント評価時に式の値が決まらない場合は，マップ式の評価結果に従って出力を制御します。

出力フォーマットでは出現数を動的に決める指定だけを実行し，コンポーネントの出現数はマップ式で制御するのが一般的です。この場合，コンポーネントの出現数を表す型コンポーネントには，評価順序遅延指定のマップ式で出力数を代入するようにします。

図 6-15 の例では，次のようにマップ式を記述します。

```
numの式：%delay %next = ARRAYSIZE(¥1@S);
Sの式：= ...@X[INDEX(0)];
```

次に，EXISTWHILE 関数又は WHILE 関数を使用して出現回数決定式を定義する方法について説明します。

入力フォーマットの場合は，EXISTWHILE 関数又は WHILE 関数を使用して，条件が満たされる間コンポーネントが出現する指定ができます。EXISTWHILE 関数を使用した場合，まだ出現していないデータの値を利用して，コンポーネントの出現の有無を決定できます。コンポーネントの出現の有無は，コンポーネントが評価される時に決定されます。

（1） EXISTWHILE 関数の使用例

EXISTWHILE 関数の使用例を次に示します。

図 6-16　EXISTWHILE 関数の使用例

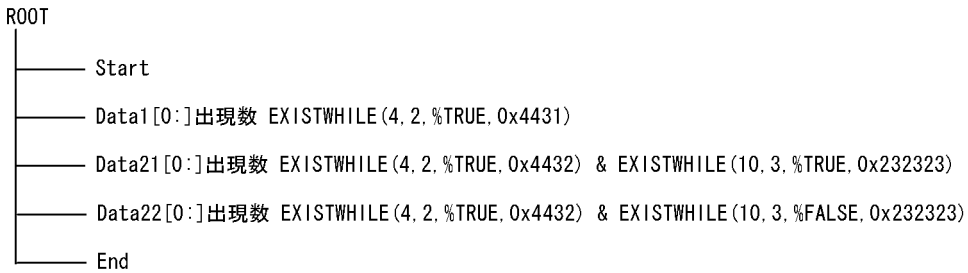
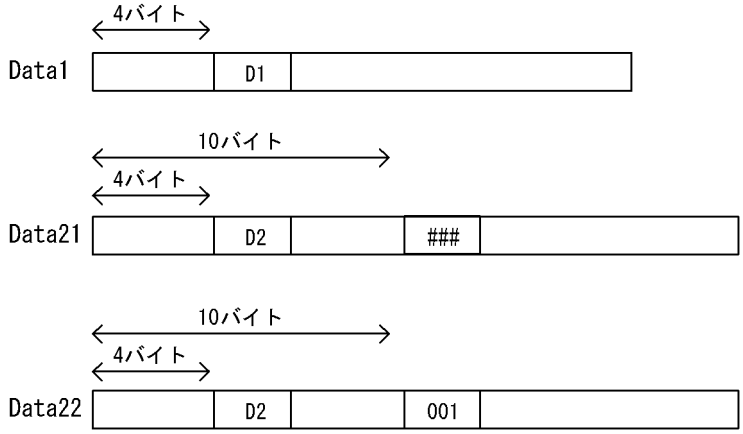


図 6-16 の例は，「Data1」「Data21」「Data22」の出現がコンポーネント中の特定位置のデータで切り替わる例です。「Data1」「Data21」「Data22」の内容は，次のようになっています。



「Data1」の出現回数決定式は，データの先頭 4 バイト目からの 2 バイトが 0x4431（"D1"）である間を Data1 の出現と見なす，という意味になります。同様に「Data21」の場合は，データの先頭 4 バイト目からの 2 バイトが 0x4432（"D2"）で，かつ 10 バイト目からの 3 バイトが 0x232323（"###"）である間を Data21 の出現と見なします。「Data22」の場合は，データの先頭 4 バイト目からの 2 バイトが 0x4432（"D2"）で，かつ 10 バイト目からの 3 バイトが 0x232323（"###"）でない場合を Data22 の出現と見なします。

EXISTWHILE 関数の条件には，固定されたデータ位置を指定する必要があります。ただし，1 コンポーネントのデータ長は任意です。

(2) WHILE 関数の使用例

WHILE 関数の使用例を次に示します。

図 6-17 WHILE 関数の使用例

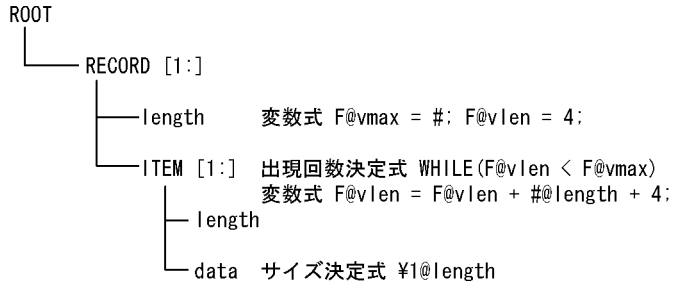


図 6-17 は，レングスタグ構造「ITEM」の可変回繰り返しが，「RECORD」直下のコンポーネント「length」で定義したサイズによって制御される例です。「RECORD」の 1 出現のサイズは，このコンポーネント「length」で決定されます。ここでは，型コンポーネント「length」のサイズを「4」とします。WHILE 関数の条件式の判定で，これまでに読み込んだデータのサイズが「RECORD」のサイズ以内である間，「RECORD」の子コンポーネント「ITEM」が出現すると見なします。

(3) EXISTWHILE 関数と WHILE 関数を組み合わせた使用例

EXISTWHILE 関数と WHILE 関数を組み合わせた使用例を次に示します。

図 6-18 EXISTWHILE 関数と WHILE 関数を組み合わせた使用例

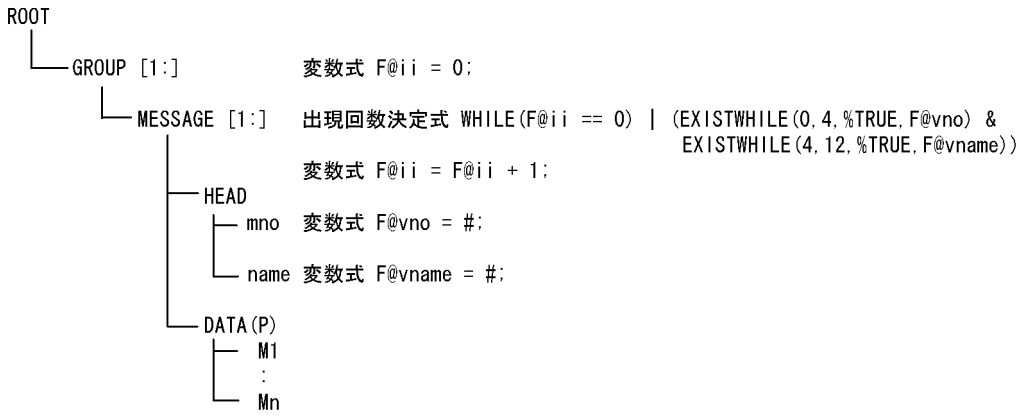


図 6-18 は，同一構成（ヘッダ部「HEAD」とメッセージ部「M1」～「Mn」）を持つ「MESSAGE」の繰り返しで，「HEAD」の情報が前回と異なる場合に，親コンポーネント「GROUP」が切り替わる例です。初回の条件は WHILE 関数，2 回目以降は EXISTWHILE 関数によって判定されます。なお，EXISTWHILE 関数の第 4 パラメタはバイト列であるため，図 6-18 の「mno」「name」が文字列の場合，不要文字削除指定

を「OFF」にしてください。不要文字削除指定については，「6.1.1(1)(e)　不要文字削除指定」を参照してください。

(4) EXISTWHILE 関数又は WHILE 関数で出現回数決定式を指定する場合の規則

EXISTWHILE 関数又は WHILE 関数で出現回数決定式を指定する場合の規則を次に示します。

- 入力フォーマットで使用する。
- 出現回数決定式を単独の項として指定するか，又は複数の EXISTWHILE 関数又は WHILE 関数を「&」又は「|」で結ぶ。

EXISTWHILE 関数で指定した位置にデータが存在しない場合は，関数の指定条件が満たされないため，コンポーネントの出現が終了したと見なされます。また，EXISTWHILE 関数の第 4 パラメタ（判定する値）又は WHILE 関数の第 1 パラメタ（条件式）による評価は，コンポーネントが出現するごとに行われます。EXISTWHILE 関数及び WHILE 関数については，「6.7.3　関数」を参照してください。

6.6.5 デフォルト式

デフォルト式は，型コンポーネントの初期値を定義する定数の代入式です。

デフォルト式を定義したコンポーネントにマップ式が定義されなかった場合に評価されます。又は，実行時オプション「-DEFMAP」が指定されていて，このコンポーネントのマップ式評価エラーになった場合，又はマップ式評価結果が %DEFAULT 定数となった場合に評価されます。

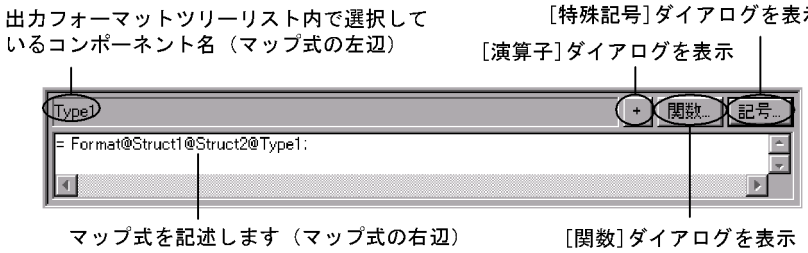
デフォルト式として記述する定数に，予約定数は使用しないでください。

デフォルト式は，マップ式と同様に個別指定ができます。詳細については，「6.6.6(8) 全体式と個別式」を参照してください。

6.6.6 マップ式

出力フォーマットのコンポーネントの値を定義するための式です。マップ式は，MDL エディタのマップビューに記述します。マップビューを次に示します。

図 6-19 マップビュー



マップ式の左辺には，出力フォーマットリストで選択しているコンポーネントの名前が表示されます。マップ式の右辺には，左辺のコンポーネントに対するマップ式を記述します。マップ式を記述するための規則について次に説明します。

- 一つのマップ式は，複数行にわたって記述できます。途中に改行が入ってもかまいません。
- 一つのコンポーネントに対して合計が 30,000 バイトまで記述できます。改行コードは 2 バイトとして数えます。
- 1 行当たりのバイト数と行数の制限はありません。
- コンポーネントが複数回出現する場合，一つのコンポーネントに対して，複数のマップ式を定義できます。ただし，一つの式の末端には，必ずセミコロン「;」を付けてください。
- 式中で使用するコンポーネント名は，グローバル名（フォーマット名から対象コンポーネントまでを「@」で連結したもの）で入力してください。
- マップ式左辺の先頭に代入式の等号「=」を記述してください（評価順序遅延指定，配列コンポーネントの複数マップ式を除きます）。ただし，コンポーネントをマップビューにドラッグ＆ドロップして初めて式を定義した場合は，自動的に等号が挿入されます。左辺が配列コンポーネントの場合は，括弧を含めたインデクス部分の後に等号を記述し，その後に右辺を記述してください。

マップ式には，複数の種類があります。

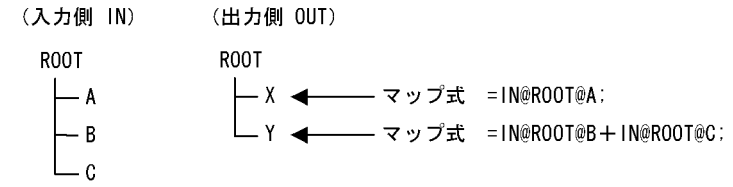
各マップ式の記述方法について次に説明します。

(1) 要素のマップ式

出力側のフォーマットの型コンポーネントに対して定義し，そのコンポーネントの値を決める式です。マップ式を記述するコンポーネントが DTD フォーマットの EMPTY 要素の場合，この式は EMPTY 要素の出力の有無を決めるために使用され，値そのものは無視されます。

次に例を示します。

図 6-20 要素のマップ式の例



X の値として A，Y の値として B+C の結果が出力されます。

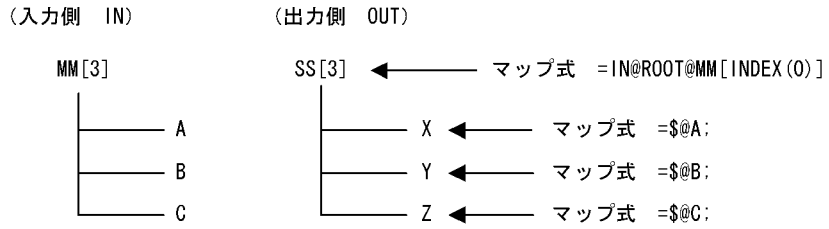
(2) 構造のマップ式

型コンポーネントだけでなく，構造コンポーネントに対してもマップ式が定義できます。CII 標準のマルチ明細のように，構造が複数回出現する場合に構造のマップ式を記述します。この場合，対応付けをする親コンポーネントの構造同士でマップ式を記述し，その子コンポーネントのマップ式に「$」を用いて親コンポーネントのマップ式を引き継ぐ形で記述します。この記述方法によって，親コンポーネントの構造間では構造としての出現を対応付け，その構造の個々の子コンポーネントの対応を決めることができます。

トランスレータでは，複数回出現する構造に対して構造のマップ式を定義し，その子コンポーネントのマップ式に「$」を用いて親コンポーネントのマップ式を引き継ぐ方法を推奨しています。ただし，無限回出現できる構造に対しては，原則として「INDEX(0)」を使用した 1 コンポーネントの代入式でなければいけません。

次の構造を例に，構造のマップ式について説明します。

図 6-21 構造のマップ式の例



入力側の構造「MM」と出力側の構造「SS」の間でマップ式を定義しています。この場合，「SS[i]@X = MM[i]@A（i = 1, 2, 3）」の意味になり，構造「MM」と構造「SS」の持つコンポーネント同士が対応付けられます。

(3) 評価順序遅延指定のマップ式

マップ式の評価順序を変更する指定です。通常のマップ式では，フォーマットに定義されたコンポーネントの順序に従い，コンポーネントが出現した時点で値を決定します。しかし，コンポーネントのサイズが次に出現するコンポーネントによって決まるような場合，次のコンポーネントが評価されていない時点では値を決定できません。

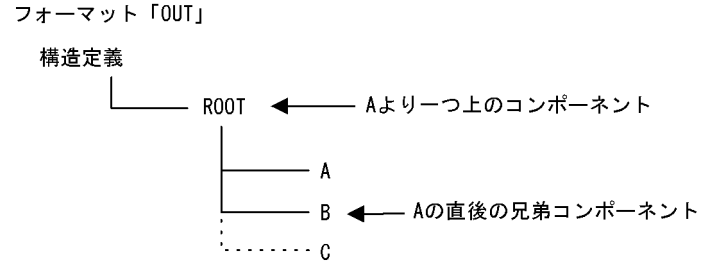
このようなとき，マップ式の評価順序を遅延させるために，評価順序遅延（%delay）を指定します。評価順序遅延の指定には，次の 2 種類があります。

- 評価するコンポーネントの直後の兄弟コンポーネント（次のコンポーネント）が評価されるまで待機するように指定（%delay %next）します。
- 評価するコンポーネントの n 階層上の親の子コンポーネントがすべて決まるまで待機するように指定（%delay n）します。評価順序遅延を指定するコンポーネントの位置を起点とした階層数「n」を「%delay」後に指定します。

評価順序遅延の指定は，マップビュー内にテキスト記入するか，又は記号として選択して指定できます。

次の構造を例に，評価順序遅延の指定について説明します。

図 6-22　評価順序遅延指定のマップ式の例



この例では，コンポーネント「B」「C」は可変長のサイズを持ち，コンポーネント「A」は構造「ROOT」のサイズを表す固定長要素とします。この例でコンポーネント「B」に対して評価順序遅延の指定をする場合の例を示します。なお，マップ式で使用する「LENGTH」は，引数で指定されたコンポーネントのサイズを計算する関数です。

- 構造「ROOT」がコンポーネント「C」を持たない場合
  コンポーネント「A」の値はコンポーネント「B」を評価すれば決まるので，コンポーネント「A」のマップ式は，直後の兄弟コンポーネントが決まったら評価する，という指定（%delay %next）になります。マップ式は，次のように記述します。

```
%delay%next = LENGTH(OUT@ROOT);
```

- 構造「ROOT」がコンポーネント「C」を持つ場合
  コンポーネント「A」の値はコンポーネント「C」まで評価しないと決まらないので，コンポーネント「A」のマップ式は，1 階層上のコンポーネントが決まったら評価する，という指定（%delay n）になります。マップ式は，次のように記述します。

```
%delay 1 = LENGTH(OUT@ROOT);
```

評価順序遅延の指定の注意事項

- 評価順序遅延の指定は，マップ式の左辺に記述してください。
- 評価順序遅延の指定は，選択構造のコンポーネントに対しては指定できません。
- 「%delay %next」を指定する場合，「%next」が指す次のコンポーネントには「%delay」を含んだマップ式は定義できません。
- 構造全体のマップ式評価を遅延させる場合は，構造に対して「%delay %next;」又は「%delay n;」と記述してマップ式を定義します。
- 親（先祖）コンポーネントの階層数を指定した場合，階層数が示すコンポーネントから，評価順序遅延マップ式を指定したコンポーネントが出現するまでの間に，選択構造のコンポーネントがあってはいけません。
- マップ式の評価順序遅延の指定は，同じ親コンポーネントを持つ兄弟コンポーネントの中で複数指定できません。
- 評価順序遅延を指定するコンポーネントのサイズは固定です（可変長要素，及び可変回数出現するコンポーネントを含む構造などには，評価順序遅延を指定してはいけません）。

（4） NULL マップ式

ある条件によってコンポーネントを出力するかどうかを指定したい場合，定数「NULL」をマップ式の値として記述できます。「NULL」を指定した場合，そのコンポーネントに対してはマップ式を定義していないことになり，そのコンポーネントは出力されません。

次に例を示します。

```
= IF(EXIST(A), "aaa", NULL);
```

この場合，コンポーネント「A」がある場合は，"aaa" を代入します。ない場合は，マップ式を記述したコンポーネントを出力しません。

Sea-NACCS EDI 電文フォーマットのコンポーネントに対して NULL マップ式が記述された場合は，スペース文字を出力します。XML 要素に対して NULL マップ式が記述された場合は，EMPTY 要素と見なし，タグだけを出力します。

NULL マップ式の注意事項

コンポーネントが省略できない場合に「NULL」をマップ式の値として記述すると変換時にエラーになります。

（5） UNDEF マップ式

コンポーネント未定義を表すマップ式を記述する場合，%UNDEF 定数を使用します。主に次の二つの用法があります。

1. 選択構造の子コンポーネントを，意図的に選択しない
2. 可変回出現コンポーネントの出力を終了させる

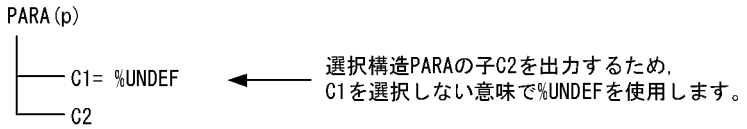
1. は，特に出力 XML フォーマットに対して有効です。XML フォーマットの場合，通常

は選択構造の順序決定式と順序決定値は設定されないため，選択構造の 2 番目以降の子コンポーネントを出力するためには，1 番目の子コンポーネントのマップ式としてエラーになるような式を記述する必要があります。このような式として UNDEF マップ式を記述できます。

なお，省略できる子コンポーネントに %UNDEF 定数をマッピングした場合，ほかに妥当な子コンポーネントがないときは，選択構造の子コンポーネントが省略されることになります。このような場合は，%UNDEF 定数をマッピングするのではなく，未使用のコンポーネントを指定して，コンポーネントを除外してください。

UNDEF マップ式の例を示します。

図 6-23 UNDEF マップ式の例



2. は，可変回出現コンポーネントの出力数を，入力コンポーネント数に関係なく決めたい場合に適用する方法です。通常，可変回出現コンポーネントには，入力コンポーネントを代入する式を記述します。さらに，INDEX 関数を使用することで，入力コンポーネントの数だけ出力するのが一般的です。しかし，場合によっては，固定数分だけ出力したい場合もあります。このような場合は，IF 文のマップ式と %UNDEF 定数を使用して，意図した数量分を出力できるようにします。

```
STRUCTURE
    X [1:] = IN@..@A[INDEX(0)];
    Y [1:] = IF (INDEX(0) <= 2, IN@..@A[INDEX(0)], %UNDEF);
```

X は入力コンポーネント A の数だけ出力されます。Y は最大 2 回出力されます。

%UNDEF は，INDEX 関数を含むパスの代入式を評価した結果，入力コンポーネントが終了したときと同じ状態と表すものです。なお，XML 要素に対して %UNDEF マップ式が記述された場合は，タグを含めてデータを出力しません。

(6) XML 属性のマップ式評価

トランスレータは，XML 属性を，構造又は型の下位にあるものとして扱います。式の評価規則は次のとおりです。

- 属性のパスは「属するコンポーネントのグローバル名 @@ 属性名」として表記します。
- パス上で属性自身はインデクスを持ちませんが（1 固定），INDEX 関数を使用できます（...@@attr[INDEX(0)] など）。

- 属性が属するコンポーネントの式評価後に，属性リストの順序（MDL エディタでの表示順）に従ってマップ式を評価します。
- 属性が属するコンポーネントの式に評価順序遅延指定がある場合，上記に従った結果として，属性の式の評価も同時に遅延されます。
- 属性のマップ式に対して，評価順序遅延指定 %delay は指定できません。
- マップ式が未定義の場合，DTD で定義されたデフォルト値を設定します。
- 属性のマップ式で使用する階層指定（¥n，INDEX 関数引数など）は，属性が属するコンポーネントを基点とします。
- # は属性が属するコンポーネントを表します。

属性のマップ式とデフォルト値の出力の関係を次に示します。

列挙型（デフォルト値有り）

- マップ式の評価結果を出力します。マップ式がない場合はデフォルト値を出力します。
- マップ式の評価結果が UNDEF の場合は，値を出力しません。
- マップ式の評価結果（UNDEF 以外）に対しては，列挙された値との合致を検証します。

#REQUIRED 属性

- マップ式の評価結果を出力します。評価結果が NULL の場合は，NULL 文字を出力します。
- 評価結果が UNDEF 又はマップ式がない場合は，結果としてエラーになります。

#IMPLIED 属性

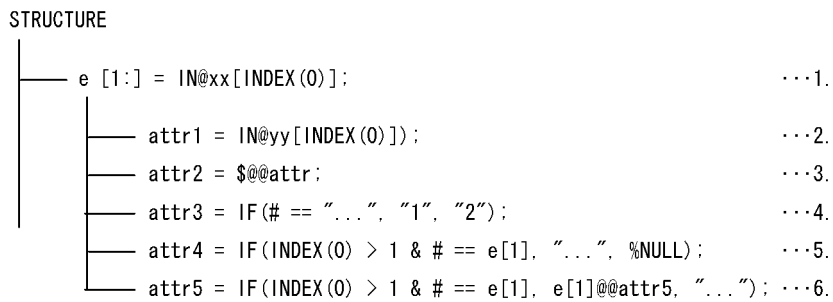
- マップ式の評価結果を出力します。NULL の場合は，NULL 文字を出力します。
- 評価結果が UNDEF 又はマップ式がない場合は，値を出力しません。

#FIXED 属性

- マップ式の評価結果を出力します。マップ式がない場合は，デフォルト値を出力します。
- マップ式の評価結果が UNDEF の場合は，値を出力しません。
- マップ式の評価結果（UNDEF 以外）に対しては，デフォルト値との合致を検証します。

属性のマップ式の例を次に示します。「e」はコンポーネント，「attri」は XML 属性を表します。

図 6-24 属性のマップ式の例



1. コンポーネントの代入式
2. 入力側のコンポーネントの値を属性値として代入，インデクスは e に対応する
3. e の入力元コンポーネント（XML として）の属性 attr の値を代入する
4. 現在の e コンポーネント値に従って属性値を決定する
5. e のコンポーネント値に従って属性値の出力有無を決定する
6. e のコンポーネント値に従って 1 番目の e の属性値と同じ値を代入する

(7) 無限回出現コンポーネントのマップ式

無限回出現コンポーネントに対しては，出力の終了が明確に判明するマップ式を記述する必要があります。通常は，INDEX(0) を含むパスの代入式で，特殊な例として %UNDEF 定数による出力終了を宣言する式です。

許容されるマップ式の形態は全体式であり，次のどれかになります。

```
= xxx[INDEX(0)];          ・・・1.
= %UNDEF;                 ・・・2.
= IF (条件, <A>, <B>);    ・・・3.
```

<A><B> は，IF 文の条件によって有限回数以内の評価と見なされる場合は任意です※。その他の場合は，1. ～ 3. でなければなりません。IF 文がネストしてもかまいません。

注※ 例えば，= IF（INDEX（0）＜ 10,（任意の式），%UNDEF）; のようなケースです。ただし，条件は INDEX（0）と定数の比較でなければなりません。

(8) 全体式と個別式

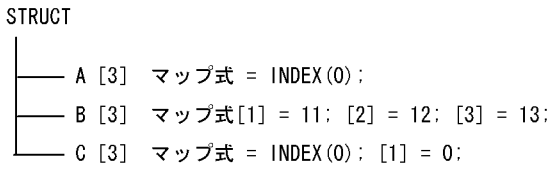
マップ式とデフォルト式には，コンポーネントの全出現に対して一つの式を定義する全体式と，個々の出現に対して式を定義する個別式があります。MDL エディタのドラッグ＆ドロップで作成される式は，全体式です。

個別式は，個々の出現ごとに別々の値や，コンポーネントを代入したい場合に使用します。全体式と個別式が両方指定された場合は，個別式が優先されます。

全体式の形式： = 式;
個別式の形式： [インデクス] = 式1; [インデクス] = 式2;...

次に例を示します。コンポーネント A には全体式，コンポーネント B には個別式，コンポーネント C には全体式と個別式を記述しています。

図 6-25 全体式と個別式の例



出力結果：
```
A [1] =  1, A[2] =  2, A[3] =  3
B [1] = 11, B[2] = 12, B[3] = 13
C [1] =  0, C[2] =  2, C[3] =  3
```

(9) 式表記と出力規則

マップ式として次のように記述した場合について説明します。

```
= IN@xxx[INDEX(0)];    ・・・1.
= %NULL;               ・・・2.
= %UNDEF;              ・・・3.
```

マップ先が XML の型コンポーネントの場合

1. 入力側 xxx の出現数分データを出力し，途中に NULL がある場合は空要素として出力します。
2. 空要素として出力します。
3. タグを含めて出力しません（コンポーネントが省略できなければ，結果としてエラーになります）。

マップ先が XML の構造コンポーネントの場合

1. 入力側 xxx の出現数分データ出力，途中の NULL は出力しません。
2. タグを含めて出力しません。
3. タグを含めて出力しません（コンポーネントが省略できなければ，結果としてエラーになります）。

マップ先が XML 以外のコンポーネントの場合

1. 入力側 xxx の出現数分データ出力します。
2. 出力しません（コンポーネントが省略できなければ，結果としてエラーになります）。
3. 出力しません（コンポーネントが省略できなければ，結果としてエラーになります）。

6.6.7 変数式

あるコンポーネントが出現した時の変数の値を更新する式です。要素のマップ式と同じ内容を定義できます。要素のマップ式については,「6.6.6(1) 要素のマップ式」を参照してください。一つのコンポーネントに対して，複数の変数式を指定できます。ただし，変数式で定義する変数は省略できないコンポーネントと同様に扱われ，出現回数は 1 回だけになります。

変数式の評価規則は次のとおりです。

- 変数式は，式を定義したコンポーネントが 1 回出現した時に評価されます。
- 変数式を定義したコンポーネントが省略された場合は，変数式は評価されません。
- 変数式は，式を定義したコンポーネントが出現した回数の分だけ評価されます。
- 変数のパスは,「フォーマット名 @ 変数名」で表記します。
- 変数自体は，インデクスを持ちません。
- 一つのコンポーネントの中では，変数式は式定義順序に従って評価されます。
- 変数式に対しては，評価順序遅延（%delay）を指定できません。ただし，変数式を定義したコンポーネントに対して評価順序遅延（%delay）が指定されている場合は，変数式の評価順序も遅延します。
- 「¥n」や INDEX 関数の引数など，変数式で使用する階層指定は，変数が所属するコンポーネントに基づいて決定されます。
- 「#」は，変数が所属するコンポーネントを表します。
- 変数式を定義したコンポーネントの親又は先祖のマップ式を引き継ぐ場合は，$ パスを使用します。

6.7 演算子及び関数

FDL エディタ及び MDL エディタで記述する式中には，四則演算，関数及び演算子を記述できます。この節では，FDL エディタ及び MDL エディタで記述する式に使用できる演算子及び関数について説明します。

また，式中で使用する定数，コンポーネント名について説明します。

6.7.1 データ型

演算子，関数の説明で使用するデータ型を次に示します。

なお，変数はグローバル名で表しますが，扱いは数値，文字列，又はバイト列の値になります。EXIST 関数や LENGTH 関数などのグローバル名としては，使用できません。

表 6-12 演算子及び関数の説明で使用するデータ型

データ型名	対応する要素の主属性	説明
i_num	整数 数値 暗黙的小数部付数値 ゾーン形式数値 パック形式数値 符号付 2 進整数 符号無 2 進整数	数値を扱うデータ型です。
	定数の主属性 • 数字列 • 浮動小数文字列 • バイト列（16 進表記数値）	
i_string	1 バイトコード文字列 2 バイトコード文字列 1 バイト／2 バイト混在文字列	文字列を扱うデータ型です。
	定数の主属性 • 文字列	
i_stream	バイトストリーム	バイト列を扱うデータ型です。
	定数の主属性 • バイト列	
i_dtm	日付 時刻 日付時刻	日付，時刻を扱うデータ型です。
i_bool	—	EXIST 関数や比較式の評価結果などのブール型です。
i_null	—	データがないことを表す型です。 マップ式で使用する定数 %NULL などに対応します。
any	（特に制限なし）	任意の型です。

データ型名	対応する要素の主属性	説明
－	グローバル名 パス名	式中で使用するコンポーネント（XML 属性を含む）の名称です。 グローバル名は，フォーマットから親と子の名称を@で連結したものです。 Format@Root@Data@Item （対象となるコンポーネント名称は Item） XML 属性を表す場合は，属性を持つコンポーネント名＋ @@ ＋属性名として表します。 Format@Root@Data@Item@@Attr （対象となる XML 属性名称は Attr） 変数を表す場合は，フォーマット名 +@+ 変数名として表します。 Format@Variable （対象となる変数は Variable） パス名は略記法も含めた総称です。

（凡例）

－：該当しません。

6.7.2 演算子

FDL 及び MDL エディタで記述する式中に使用できる演算子について説明します。演算子の書式と意味を次に示します。二項演算子の結合規則はすべて左結合です。

表 6-13 演算子一覧

名前	書式	結果型	意味	優先順位
|	i_bool1 | i_bool2	i_bool	i_bool1 と i_bool2 のどちらかが TRUE のとき TRUE を，それ以外は FALSE を返す。	7
&	i_bool1 & i_bool2	i_bool	i_bool1 と i_bool2 が共に TRUE のとき TRUE を，それ以外は FALSE を返す。	6
==	i_num1 == i_num2	i_bool	i_num1 と i_num2 が 10 進数の値として等しいとき TRUE を，それ以外は FALSE を返す。	5
	i_string1 == i_string2		1. 文字コード種別が同じ場合： 長さ及びバイト内容が等しいとき TRUE を，それ以外は FALSE を返す。 2. 文字コード種別が異なる場合： i_string2 を i_string1 の文字コードに変換してから 1 を適用する。	
	i_stream1 == i_stream2		長さとバイト内容が等しいとき TRUE を，それ以外は FALSE を返す。	

名前	書式	結果型	意味	優先順位
	i_null1 == i_null2		i_null1 と i_null2 が共に i_null 型のとき TRUE を，どちらか一方だけが i_null 型のとき FALSE を返す。	
!=	i_num1 != i_num2	i_bool	i_num1 と i_num2 が 10 進数の値として等しいとき FALSE を，それ以外は TRUE を返す。	5
	i_string1 != i_string2		1. 文字コード種別が同じ場合： 長さ及びバイト内容が等しいとき FALSE を，それ以外は TRUE を返す。 2. 文字コード種別が異なる場合： i_string2 を i_string1 の文字コードに変換してから 1 を適用する。	
	i_stream1 != i_stream2		長さとバイト内容が等しいとき FALSE を，それ以外は TRUE を返す。	
	i_null1 != i_null2		i_null1 と i_null2 が共に i_null 型のとき FALSE を，どちらか一方だけが i_null 型のとき TRUE を返す。	
<	i_num1 < i_num2	i_bool	i_num1 < i_num2 のとき TRUE を，それ以外は FALSE を返す。	5
<=	i_num1 <= i_num2	i_bool	i_num1 <= i_num2 のとき TRUE を，それ以外は FALSE を返す。	5
>	i_num1 > i_num2	i_bool	i_num1 > i_num2 のとき TRUE を，それ以外は FALSE を返す。	5
>=	i_num1 >= i_num2	i_bool	i_num1 >= i_num2 のとき TRUE を，それ以外は FALSE を返す。	5
+	i_num1 + i_ num2	i_num	i_num1 に i_num2 を加算した値を返す。	4
-	i_num1 - i_num2	i_num	i_num1 から i_num2 を減算した値を返す。	4
*	i_num1 * i_num2	i_num	i_num1 に i_num2 を乗算した値を返す。	3
/	i_num1 / i_num2	i_num	i_num1 を i_num2 で除算した値を返す。	3
単項 +	+i_num1	i_num	i_num1 に 1 を乗算した値を返す。	2
単項 -	-i_num1	i_num	i_num1 に -1 を乗算した値を返す。	2
!	! i_bool	i_bool	i_bool が TRUE のとき FALSE を，FALSE のとき TRUE を返す。	2
()	(any1)	any1 の型	any1 の評価結果を返す（優先順位※）。	1

注※

優先順位とは，式中で演算子が複数使用されたときの優先順位を示します。

表中の書式の第 1 子と第 2 子を逆に指定することもできます。また，書式の i_num 型の代わりに，i_string，i_stream，i_dtm も指定できます。この場合は，i_num 型に変換してから演算子を適用します。変換の詳細は，「6.5.1　異なる属性型同士での変換」の「表 6-11　異なる分類の属性の型同士での変換可否」を参照してください。ただし，「==」「!=」演算の i_string 型同士や，i_stream 型同士が適用される場合を除きます。

6.7.3　関数

FDL エディタ及び MDL エディタの式中に使用できる関数について説明します。FDL エディタ及び MDL エディタで記述する式中には，Interschema が標準で提供している関数以外に，ユーザが定義した関数を組み込んで使用できます。ここでは，標準で提供している関数について説明します。ユーザが定義した関数を組み込む方法については，「11.　ユーザ組み込み関数」を参照してください。

FDL エディタ及び MDL エディタで記述する式中で使用できる関数の一覧を次に示します。

表 6-14　関数一覧

関数種別	関数の書式	戻り値型	説明
数字操作	ROUND (i_num1, 整数定数 i)	i_num	i_num1 を指定桁で四捨五入した値を返します。 • i >= 0 のとき，小数点以下第 i+1 桁を四捨五入 • i < 0 のとき，整数部第 -i 桁を四捨五入
	FLOOR (i_num1, 整数定数 i)	i_num	i_num1 を指定桁で切り捨てた値を返します。 • i >= 0 のとき，小数点以下第 i+1 桁を切り捨てる • i < 0 のとき，整数部第 -i 桁を切り捨てる
	CEIL (i_num1, 整数定数 i)	i_num	i_num1 を指定桁で切り上げた値を返します。 • i >= 0 のとき，小数点以下第 i+1 桁を切り上げる • i < 0 のとき，整数部第 -i 桁を切り上げる
	SUM (i_num1, ...) ※1	i_num	i_num1... の和を返す。

関数種別	関数の書式	戻り値型	説明
文字列操作	LEFT (i_string1, i_num1)	i_string ※2	i_string1 の左端（先頭）から INT(i_num1) 文字目までの文字列を取り出します。i_string1 の文字数が INT(i_num1) 文字より小さい場合，i_string1 全体を返します。
	RIGHT (i_string1, i_num1)		i_string1 の右端（末尾）から INT(i_num1) 文字目までの文字列を取り出します。i_string1 の文字数が INT(i_num1) 文字より小さい場合，i_string1 全体を返します。
	MID (i_string1, i_num1, i_num2)		i_string1 の左端（先頭）の INT(i_num1) 文字目から INT(i_num2) 文字目までの文字列を取り出します。INT(i_num1) が 0 の場合，文字列の先頭から INT(i_num2) 文字目までの文字列を取り出します。INT(i_num2) が 0 の場合，INT(i_num1) 文字目以降のすべての文字列を取り出します。
	STRLEN (i_string1)	i_num	i_string1 の文字数を返します。
	ADDSTRING (i_string1, ...) ※1	i_string	i_string1... を指定された順序で連結します。※3
バイト列操作	ADDSTREAM (i_stream1, ...) ※1	i_stream	i_stream1... を指定された順序で連結します。
	CUTSTREAM (i_stream1, i_num1. i_num2)	i_stream ※2	i_stream1 の先頭 INT(i_num1) バイト目から INT(i_num2) バイト目までのバイト列を取り出します。INT(i_num1) が 0 の場合，バイト列先頭から INT(i_num2) バイトのバイト列を取り出します。INT(i_num2) が 0 の場合，INT(i_num1) バイト目以降のすべてのバイト列を取り出します。
日時操作	CURRENTDATE ( )	i_dtm	現在の日付を返します。
	CURRENTTIME ( )		現在の時刻を返します。
	CURRENTDTM ( )		現在の日付時刻を返します。
	SETDATETIME (CC,YY,MM,DD,hh,mm,ss,zz) CC ～ ss：i_num zz：i_string or i_null		日付時刻型の値を設定します。パラメタの意味は，次のとおりです。 CC ＝西暦年上位 2 桁，YY ＝西暦年下位 2 桁，MM ＝月，DD ＝日，hh ＝時，mm ＝分，ss ＝秒，zz ＝ゾーン ゾーン部なしとする場合は，zz に %NULL を指定します。

関数種別	関数の書式	戻り値型	説明
	GETDATETIME (整数定数 i, i_dtm)	i_num	整数定数 i の値に応じて次のパート値を返します。 i = 0　CCYY（西暦年） i = 1　CC　（西暦年上位 2 桁） i = 2　YY　（年） i = 3　MM（月） i = 4　DD　（日） i = 5　hh　（時） i = 6　mm（分） i = 7　ss　（秒）
	GETDTMZONE (i_dtm)	i_string	日付時刻型のゾーン部文字列を返す。ゾーン部がない場合は i_null を返します。

関数種別	関数の書式	戻り値型	説明
型変換	TOSTRING (i_num1, i_string1)	i_string	i_num1 を書式文字列 i_string1 に従って i_string 型に変換します。変換結果の文字列の文字コードは JIS8 になります。 書式文字列の形式は次のとおりです。形式に従って 1 バイト文字で指定します。[ ] 内は省略できます。符号は，先頭又は末尾のどちらかにだけ指定できます。 書式 1 形式： [I] [符号] [整数部桁数] [.[小数部桁数]] [符号] • 符号： 「*」は必ず付けます。「-」は負の場合だけ付けます。 桁数の指定がない場合は有効数字だけ出力します。指定された桁数が実際の桁数より小さい場合，整数部は上位桁，小数部は下位桁が切り捨てられます（エラーにはなりません）。指定された桁数が実際の桁数より大きい場合，不足桁分には 0 を補います。 桁数に 0 を指定した場合は，その整数部又は小数部の数値を出力しません（0 も出力しません）。符号指定がない場合は負の値であっても符号を付けません。 書式の先頭に「I」を指定した場合は，小数点文字を出力しません。書式の先頭に「I」を指定する場合は，小数部桁数の指定は省略できません。 前負符号，整数部 2 桁，小数部 1 桁を指定する書式文字列は，“ -2.1 ” となります。 書式 2 形式：H[0][X] 又は h[0][x] • 0： 偶数バイトで出力します。奇数バイトになるデータの場合は，前に「0」が付加されます。 • x ／ X： 先頭に「0x」が付加されます。 • H： 英大文字で文字列化されます。 • h： 英小文字で文字列化されます。 16 進表記文字列に変換します。小数部及び負符号は無視されます。
	HEXSTRTONUM (i_string1)	i_num	16 進表記文字列 i_string1 を i_num 型に変換します。文字列先頭の「0x」の有無，及び英大文字 / 英小文字の使用は任意です。また，文字列は奇数バイトでも使用できます。

関数種別	関数の書式	戻り値型	説明
	HEXSTRTOSTREAM (i_string1)	i_stream	16 進表記文字列 i_string1 を i_stream 型に変換します。文字列先頭の「0x」の有無，及び英大文字 / 英小文字の使用は任意です。また，文字列は奇数バイトでも使用できます。
	STREAMTOHEXSTR (i_stream1)	i_string	i_stream1 を 16 進表記文字列に変換します。文字列先頭の「0x」は付加されません。英小文字で文字列化されます。変換結果文字列の文字コードは JIS7 です。出力文字列は偶数バイトになります。
条件	IF (条件式 , any1, any2)	any1 又は any2	条件式の評価結果が TRUE なら any1，FALSE なら any2 を評価して返します。戻り値型は評価結果に依存します。
	IFEXIST (グローバル名 , any)	any	グローバル名が指すコンポーネントがあれば該当するコンポーネント，なければ any を返します。 対象が型要素の場合は，値がある場合にあると見なします。対象が構造の場合は，子コンポーネント以下の要素のどれかに値があればあると見なします。セパレータしか現れない場合は，ないと見なします。 コンポーネントがあるかないかの判定は，「その他」の EXIST 関数と同じです。
XML 操作	GETTAGNAME (グローバル名)	i_string	グローバル名が XML 要素を表す場合，対応する XML タグ名を返します。グローバル名が XML 属性を表す場合，該当する属性名を返します。 その他の場合は i_null を返します。
	GETANYTEXT (ANY 型グローバル名 , 整数定数 i)	i_string	ANY 型のコンポーネントのデータから，タグ間テキストだけを抽出して連結したデータを返します。指定したコンポーネントが ANY 型でないなどの不正時は i_null を返します。最上位要素（ANY 型自身）のテキストデータの white space の扱いは，実行時オプション「-XWS」に従った結果に対して処理します。 テキスト連結方法は，整数定数 i の指定に従います。 • i =0 の場合 タグ間テキストをそのまま連結します。 • i =1 の場合 タグごとに前後の white space を削除して連結します。 • i =2 の場合 文字列中の white space をスペース文字に置換し，連結後の文字列前後の white space を削除します。また，連続するスペースを 1 スペース文字に置換します。

関数種別	関数の書式	戻り値型	説明
	EXISTXML (グローバル名)	i_bool	評価済みのコンポーネントがあれば TRUE，なければ FALSE を返します。 対象が型要素の場合は，値がある場合にあると見なします。対象が構造の場合は，子コンポーネント以下の要素のどれかに値があればあると見なします。セパレータしか現れない場合は，ないと見なします。対象が XML 要素又は XML 属性の場合は，タグ又は属性名が出現すれば，データがなくてもあると見なします（空要素をあると見なします）。 EXISTXML 関数は，XML 要素又は XML 属性に対しての判定以外は，「その他」の EXIST 関数と同じです。
	EXISTCOUNTXML (グローバル名 , ...) ※1	i_num	引数すべてに EXISTXML 関数を適用して，TRUE になった数を返します。 対象が XML 要素又は XML 属性の場合は，タグ又は属性名が出現すれば，データがなくてもあると見なします（空要素をあると見なします）。 EXISTCOUNTXML 関数は，XML 要素又は XML 属性に対しての判定以外は，「その他」の EXISTCOUNT 関数と同じです。
	ARRAYREALSIZEXML (グローバル名)	i_num	評価済みのコンポーネントの配列要素数を返します（空の要素はカウントしません，XML の空要素はカウントします）。 対象が XML 要素又は XML 属性の場合は，タグ又は属性名が出現すれば，データがなくてもあると見なします（空要素をあると見なします）。 ARRAYREALSIZEXML 関数は，XML 要素又は XML 属性に対しての判定以外は，「その他」の ARRAYREALSIZE 関数と同じです。
	GETCOMPXML (整数定数 i, グローバル名 , ...) ※1 整数定数は 1 以上	any	グローバル名で指定されたコンポーネントの中から，整数定数 i 番目に出現したコンポーネントを返します。 対象が XML 要素又は XML 属性の場合は，タグ又は属性名が出現すれば，データがなくてもあると見なします（空要素をあると見なします）。 GETCOMPXML 関数は，XML 要素又は XML 属性に対しての判定以外は，「その他」の GETCOMP 関数と同じです。
	GETCOMPIFXML (基点パス名 , 条件式 , 整数定数 i, 連結パス名 , ...) ※1 整数定数は 1 以上	any	連結パス名で指定されたコンポーネントで指定条件式を満たすもののうち，整数定数 i 番目に出現したコンポーネントを返します。 対象が XML 要素又は XML 属性の場合は，タグ又は属性名が出現すれば，データがなくてもあると見なします（空要素をあると見なします）。 GETCOMPIFXML 関数は，XML 要素又は XML 属性に対しての判定以外は，「その他」の GETCOMPIF 関数と同じです。

関数種別	関数の書式	戻り値型	説明
	IFEXISTXML (グローバル名 , any)	any	グローバル名が指すコンポーネントがあれば該当するコンポーネント，なければ any を返します。 対象が型要素の場合は，値がある場合にあると見なします。対象が構造の場合は，子コンポーネント以下の要素のどれかに値があればあると見なします。セパレータしか現れない場合は，ないと見なします。対象が XML 要素又は XML 属性の場合は，タグ又は属性名が出現すれば，データがなくてもあると見なします（空要素をあると見なします）。 コンポーネントがあるかないかの判定は，「XML 操作」の EXISTXML 関数と同じです。
特殊	ORDEROFFSET (i_num1, i_num2)	—	現在の位置から i_num1 相対オフセット位置にある i_num2 長のバイト列を，順序決定式の値とします。 順序決定式を定義する場合だけ使用できます。
	EXISTWHILE (i_num1, i_num2, i_bool1, i_stream1)	—	現在の位置から i_num1 相対オフセット位置にある i_num2 長のバイト列を，関数引数として指定された i_stream1 バイト列と比較します。 条件 i_bool1 が %TRUE の場合，比較した結果が一致する間，コンポーネントが出現すると見なします。 条件 i_bool1 が %FALSE の場合は，比較した結果が不一致の間，コンポーネントが出現すると見なします。 第 4 パラメタ i_stream1 は，コンポーネントの出現を判定するごとに毎回評価されます。 出現回数決定式を定義する場合だけ使用できます。
	WHILE (条件式)	—	条件式が満たされる間，コンポーネントが出現すると見なします。条件式は，コンポーネントの出現を判定するごとに毎回評価されます。 出現回数決定式を定義する場合だけ使用できます。
その他	LENGTH (グローバル名)	i_num	グローバル名が指すコンポーネントのバイト長を返します。ただし，対象が XML 要素（含む属性）の場合は，常に 0 を返します。 グローバル名で指定されたコンポーネントの子孫がグループ単位出力指定で処理済みの場合，そのコンポーネント分のサイズは 0 として計算します。
	INDEX (整数定数) 整数定数は 0 以上	i_num	対象コンポーネント（現在評価中の式が定義されているコンポーネント）の整数定数で指定された分だけ上の階層にあるコンポーネントの，現在のインデクスを返します。 構造に対する評価規則中の INDEX 関数は，その構造がコンポーネントとして使われている場所を 0 として適用します。

関数種別	関数の書式	戻り値型	説明
	ARRAYSIZE (グローバル名)	i_num	評価済みのコンポーネントの配列要素数を返します（空の要素もカウントします）。
	ARRAYREALSIZE (グローバル名)	i_num	評価済みのコンポーネントの配列要素数を返します（空の要素はカウントしません）。
	EXIST (グローバル名)	i_bool	評価済みのコンポーネントがあれば TRUE，なければ FALSE を返します。 対象が型要素の場合は，値がある場合にあると見なします。対象が構造の場合は，子コンポーネント以下の要素のどれかに値があればあると見なします。セパレータしか現れない場合は，ないと見なします。 対象が XML 要素で，データを持たない場合（EMPTY 要素などタグだけ出現するもの）は，FALSE を返します。
	EXISTCOUNT (グローバル名 ,...) ※1	i_num	引数すべてに EXIST 関数を適用して，TRUE になった数を返します。
	VALUEMAP(入力側要素グローバル名 , 出力側要素グローバル名 , { 入力側項目番号 , 出力側項目番号 }, ...)	出力要素の型	値定義を持つ要素間で，値を対応付けます。対応付ける値の項目番号（値定義での出現順）を組にして複数指定します。項目番号「0」は，値定義に現れない値を意味します。
	GETCOUNT( )	i_num	システムが持っているカウンタの値を返します。FDL 又は MDL の中でカウンタが指定されていない場合は，何も返しません。
	GETCOMP(整数定数 i, グローバル名 , ...) ※1 整数定数は 1 以上	any	グローバル名で指定されたコンポーネントの中から，整数定数 i 番目に出現したコンポーネントを返します。GETCOMP 関数は，ほかの関数（GETCOMPIF 関数 ,GETCOMPIFXML 関数を除く）の引数のグローバル名として使用できます。
	GETCOMPIF(基点パス名 , 条件式 , 整数定数 i, 連結パス名 , ...) ※1 整数定数は 1 以上	any	連結パス名で指定されたコンポーネントで指定条件式を満たすもののうち，整数定数 i 番目に出現したコンポーネントを返します。 GETCOMPIF 関数は GETCOMPIF 関数，GETCOMPIFXML 関数 を除いたほかの関数の引数のグローバル名として使用できます。また，GETCOMPIF 関数の引数に GETCOMP 関数，GETCOMPXML 関数，GETCOMPIF 関数，及び GETCOMPIFXML 関数は使用できません。

（凡例）

－：戻り値はありません。

注※1

引数のインデクスに「*」を使用できる関数です。

注※2
取り出す文字列（又はバイト列）がない場合はi_null型を返します。なお，文字列（又はバイト列）の先頭の文字数（又はバイト数）は1です。

注※3
文字コードが異なる文字列を連結する場合，文字セットとして大きい側の文字コードへ変換します。

VALUEMAP関数，GETCOMP関数，及びGETCOMPIF関数については，例を挙げて次に説明します。

（1）VALUEMAP関数

VALUEMAP関数を使用した，要素の値同士のマッピング例を次に示します。

入力側要素：IN@A
値リスト：{"AAA", "BBB", "CCC"}値定義のエラー検証なし

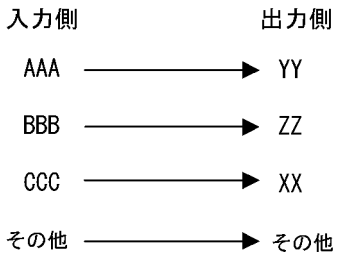
出力側要素：OUT@X
値リスト：{"XX", "YY", "ZZ"}値定義のエラー検証なし

マップ式を次のように記述します。

```
= VALUEMAP(IN@A, #, 1, 2, 2, 3, 3, 1, 0, 0);
```

この場合のマッピング関係を次に示します。「その他」は，値定義されていない値を意味し，入力された値をそのまま出力する指定になります。

図6-26　VALUEMAP関数を使用した値のマッピング例

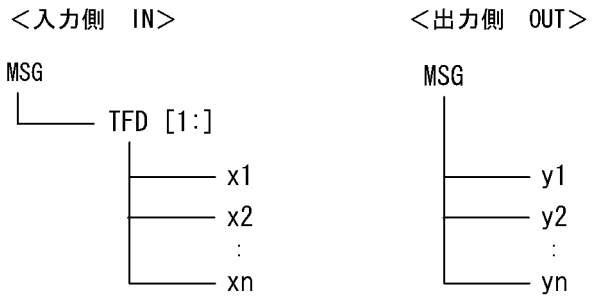


（2）GETCOMP関数

GETCOMP関数は，入力側にある選択構造の子コンポーネントで，何番目に出現するか分からないコンポーネントを取り出す場合などに使用します。

GETCOMPの例を次に説明します。

図 6-27 GETCOMP 関数を使用した例



入力側フォーマット「IN」は，選択構造「TFD」の要素「x1」から「xn」が順不同で現れ，出力側フォーマット「OUT」は，要素「y1」から「yn」を順番に出力すると仮定します。また，入力側の要素「xi」を出力側の要素「yi」へマッピングすると仮定します。

この場合，要素「yi」に対するマップ式は次のようになります。

```
= GETCOMP( 1, IN@...@MSG@TFD[*]@xi) ;
```

入力側の構造「TFD」の子コンポーネントのうち，1 番目に現れた「xi」を「yi」へ代入する」意味になります。

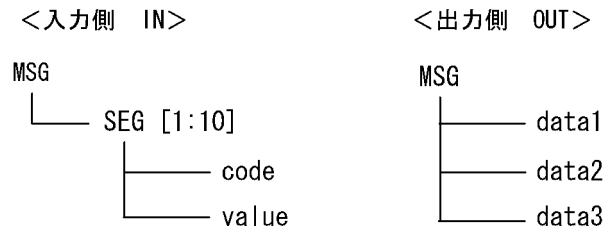
(3) GETCOMPIF 関数

GETCOMPIF 関数は，GETCOMP 関数によって取り出されるコンポーネントに対する条件を追加したものです。例えば，入力側で繰り返し出現する構造コンポーネントがコードを表す子コンポーネントと値を表すコンポーネントを持つ場合に，その構造コンポーネントが特定のコードを持つときの値（要素）を取り出すために使用します。

基点パス名には検索対象のコンポーネントを指定します。連結パス名は実際に取り出す対象のコンポーネントを指定します。条件式内のパス名及び連結パス名では，基点パスで指定したコンポーネントの子に対して，基点パス部分を「&」とし，パス名を「&@xxx」と記述できます。基点パスコンポーネントそのものを表す場合は「&&」と指定します。基点パス名及び連結パス名ではインデクスに「*」を使用できますが，条件式内では使用できません。

GETCOMPIF 関数の使用例を次に示します。

図 6-28 GETCOMPIF 関数を使用した例



例のように，入力側フォーマット「IN」の構造「SEG」から，code が「1」であるときの value を OUT フォーマットの data1 へ，code が「2」のときの value を OUT フォーマットの data2 へ出力する場合，data1 及び data2 のマップ式は次のようになります。

```
data1 = GETCOMPIF(IN@...@MSG@SEG[*], &@code == 1, 1, &@value);
data2 = GETCOMPIF(IN@...@MSG@SEG[*], &@code == 2, 1, &@value);
```

6.7.4 定数

式の中で固定の値として使用するものです。定数には，文字列，数字列，浮動小数文字列，バイト列，予約定数があります。各定数について次に説明します。

(1) 文字列

文字列をダブルクォーテーション「"」で囲んでください。次に例を示します。

"ABC 漢字 "

なお，文字列中にダブルクォーテーション「"」を使用する場合，次のように「"」を二つ連続で記述してください。

指定したい文字列：a"b

実際に指定する文字列："a""b"

(2) 数字列

0 ～ 9 の数字で構成されます。先頭に 0 があってもかまいません。負の値を表す場合は先頭に単項演算子の「-」を使用します。また，先頭に「0x」を付けた 16 進数字で記述することもできます。ただし，符号なしの扱いで，表現はビッグエンディアンの形式と見なします。

次に例を示します。

01234567890

-1234567890

0x0100 （＝ 256）

(3) 浮動小数文字列

「数字」「小数点」「符号」「指数記号」から構成されます。小数部を持つ数字や，指数表現のある数値を表します。指数記号は，「E」又は「e」だけを使用できます。負の値を表す場合は数字列と同じように，単項演算子の「-」を使用します。次に例を示します。

123.45

-6789.0

1.2E+5

(4) バイト列

先頭に「0x」を付けた 16 進数字です。記述する値は，フォーマットのエンディアンの指定に従う必要があります。また，複数バイトを占める定数の場合は，バイト数分指定してください。次に例を示します。

0xFEFF003C003F

(5) 予約定数

NULL（又は %NULL）
: 「NULL」と「%NULL」の意味はどちらも同じです。
マップ式内だけで使用できます。例外として，Sea-NACCS のフォーマットとして，デフォルト式に記述できます（スペース文字の埋め込みの意味）。
DTD フォーマットの型コンポーネントに対して「NULL」又は「%NULL」を指定した場合，そのコンポーネントは，XML のタグだけを出力します。その他のコンポーネントに対しては，マップ式を定義していないことになります。

%TRUE
: BOOLEAN 定数を表します。真を表します。

%FALSE
: BOOLEAN 定数を表します。偽を表します。

%DEFAULT
: 型コンポーネントのデフォルト式を表します。マップ式内でだけ使用できます。
マップ式の評価結果がこの定数になった場合，デフォルト式の値が出力されます。

%UNDEF
: コンポーネント未定義を表します。マップ式内でだけ使用できます。
DTD フォーマットの型コンポーネントに対して「%UNDEF」を指定した場合，XML のタグを含めて出力しません。その他のコンポーネントの場合は，「NULL」が指定された場合と同じです。
「%UNDEF」は「NULL」と似た機能を持ちますが，「NULL」はコンポーネントの 1 出現に対する未定義を表し，「%UNDEF」はコンポーネントの出現終了を表しま

す。無限回出現コンポーネントの出力を終了させる場合などに使用できます。

6.7.5 コンポーネント名

式中に記述するコンポーネント名について説明します。

(1) グローバル名

コンポーネント名はグローバル名で指定します。グローバル名とは，フォーマット名から対象コンポーネントまでを「@」で連結したものです。コンポーネントが XML 属性を表す場合，属性コンポーネントの前は，「@@」でつなぎます。また，変数名を表す場合は，フォーマット名と変数名を「@」でつなぎます。次に，グローバル名の例を示します。

グローバル名の例（XML 属性を表す場合）：

```
EXAMPLE@ROOT@A@@B
```

XML 属性「B」のグローバル名を表しています。グローバル名から，フォーマット名は「EXAMPLE」，ルート構造「ROOT」の子のコンポーネントが「A」であることが分かります。

グローバル名の例（変数を表す場合）：

```
EXAMPLE@VARIABLE
```

変数「VARIABLE」のグローバル名を表しています。グローバル名から，フォーマット名は「EXAMPLE」であることが分かります。

FDL ファイルのフォーマット内では，同一のコンポーネント名を共有できるので，一つのコンポーネントは構造中の異なる複数のコンポーネントの子のコンポーネントとしても使用できます。ただし，MDL ファイルでは，すべてのコンポーネントは，それぞれ独自でなければなりません。このため，FDL ファイル内で共用されていたコンポーネント名は，MDL ファイルに読み込むと自動的に，フォーマット内で一意の名称に付け替えられます。名称が付け替えられるときは，そのコンポーネントの MDL 内での出現位置が基になります。名称が付け替えられても，構造の内容は変わりません。FDL ファイル内のコンポーネント名を「ローカル名」，MDL ファイル内のコンポーネント名を「グローバル名」と呼びます。

なお，式中で指定できるコンポーネントは，式を記述したコンポーネントを評価する以前に内容が決定されているものでなければなりません。まだ出現していないコンポーネントの値を参照した場合は，エラーになるので注意してください。

(2) 要素名称の規則

要素名称とは，FDL ファイル内のコンポーネントの名称です。ローカル名とも呼びます。

ここでは，FDL で定義する要素名称の規則について説明します。要素名称として使用できる文字は，次のとおりです。

- １バイトの片仮名
- ２バイトの数字文字
- XML 1.0 で名前として使用できる文字※

  注※
  アルファベット，平仮名，片仮名（２バイト），数字，結合文字（アクセント記号，ウムラウトなど），「_」（アンダースコア），「:」（コロン），「.」（ピリオド），「-」（ハイフン）のことです。

ただし，２バイト文字はシフト JIS で表示できるものに限ります。また，XML 1.0 では，先頭文字として使用できない文字を定めていますが，Interschema では区別しません。

定義できる名前の長さは，フォーマット名は 256 バイト以内，その他の名前は 4,096 バイト以内です。

特殊な名前や，数字で始まったり，記号文字が含まれたりする名前

Interschema が自動的に生成する XML ダミー構造などの名称には，特殊文字を使用する場合もあります。このような特殊な名前や，数字で始まったり，記号文字が含まれたりする名前に対しては，名前の前後にシングルクォーテーションを付けて表示します。条件式やマップ式などで名称を記述する場合にも，シングルクォーテーションが必要です。

特殊な名前の例：

```
'0123'
'ABC-DEF'
```

特殊な名前を含むコンポーネントのグローバル名の例：

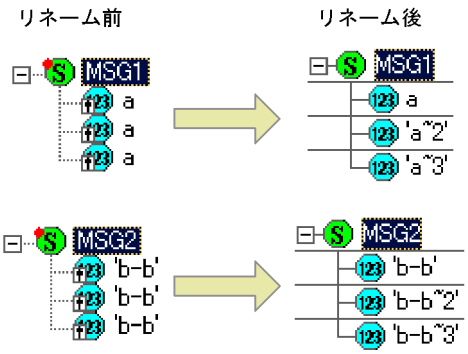
```
Format@Root@'0123'
Format@Root@Msg@'ABC-DEF'@element
```

シングルクォーテーションを必要としない名前は，1 バイト英数字，1 バイト片仮名，アンダースコア，及び 2 バイト文字で構成される名前です。ただし，先頭は 1 バイト数字文字以外の名前となります。

リネーム前後の名前

FDL では同名の兄弟コンポーネントが定義できます。これらは MDL へ取り込む時点で，ローカル名に相当する部分がリネームされ，ユニークな名称に置き換わります。リネーム後の名称は，上記に示すような特殊な名前になります。
リネームでは，ローカル名に「~n」が付加されます。n は兄弟コンポーネント間の順序番号（n>1）です。ただし，同名の１番目のコンポーネントには「~n」が付加されません。また，リネーム前の名称が特殊な名前の場合は，シングルクォーテーションの内側に「~n」が付加されます。

リネームの例：



(3) コンポーネント名の略記方法

コンポーネント名には，次のような略記方法があります。

「#」

式が記述されているコンポーネント，つまり自コンポーネントを表します。ただし，式を記述するコンポーネントがXML属性の場合は，その属性が属するコンポーネントを表します。変数式の場合は，式を定義したコンポーネントを表します。

条件式で使用した例：

```
# == 1（自身の値が1）
```

「¥n」

自コンポーネントの親（先祖）のコンポーネントを表します。nは，自コンポーネントのn階層上のコンポーネントを表す番号です。自コンポーネントの親コンポーネントは1階層上のコンポーネントなので「¥1」と表します。ただし，式を記述するコンポーネントがXML属性の場合は，属性が属するコンポーネントの階層を0として数えます。XML属性自身に対しては，階層を数えません。変数式の場合は，式を定義したコンポーネントの階層を0として数えます。
レングスタグ構造でデータ長を持つ要素を表す場合など，親コンポーネントの構造を示すときに使用できます。

サイズ決定式で使用した例：

```
¥1@LEN　（自身のデータサイズは，兄弟コンポーネントのLENの値）
```

「$」

マップ式及び変数式の中でだけ記述できます。マップ式で使用した場合，式が記述されているコンポーネントの親（先祖）のコンポーネントで，マップ式の評価結果が表すコンポーネントを表し，親（先祖）のコンポーネントのマップ式の評価結果を引き継ぐコンポーネントであることを示します。変数式で使用した場合は，式を定義したコンポーネントの親（先祖）のマップ式を引き継ぐことを示します。
構造コンポーネントのマップ式では，そのコンポーネントを出力するという意味として「=$;」の形でも使用できます。ただし，親（先祖）のコンポーネントにマップ

式が定義されていなければなりません。

マップ式で使用した例：

```
= $@data ;（親コンポーネントにマッピングされたコンポーネントの子コンポーネントに位置するdataを代入する）
```

「&」

GETCOMPIF 関数又は GETCOMPIFXML 関数のパラメタの中で使用し，第 1 パラメタの基点パスに連結するパス名を表します。詳細は，「6.7.3　関数」を参照してください。

6.7.6　インデクスの指定

コンポーネント名を指定する場合，又はルート構造から自コンポーネントまでに複数回出現するコンポーネントがある場合は，式中でインデクスを指定する必要があります。ただし，変数はインデクスを持っていないため，変数のグローバル名に対してはインデクスを指定できません。

インデクスの指定方法には次の 4 種類があります。

- 固定の値を指定して何番目に出現するコンポーネントかを示します。
- 算術式などを使用して特定します（例えば，INDEX 関数による指定）。
- ワイルドカード「*」を使用します。ワイルドカードは，複数回出現するコンポーネントの出現すべてを指定する意味になります。
  なお，「*」が指定できる関数は限定されています。Interschema が提供する関数については，「6.7.3　関数」の「表 6-14　関数一覧」を参照してください。ユーザ組み込み関数の場合は，可変個の引数を持つ関数で「*」が使用できます。
- 「?」を使用します。「?」は，マップ式が記述されたコンポーネントの現在の出現のインデクスを表します。複数回出現するコンポーネントのマップ式で使用すると，入力側のコンポーネントと式を定義している出力側のコンポーネントを 1 対 1 に同じ順序で対応付けます。現在の出現インデクスを表します。INDEX 関数で「INDEX(0)」と記述するのと同じ意味です。マップ式の左辺では使用できません。

7 ユティリティ

この章では，Interschema が提供しているユティリティプログラムについて説明します。

7.1 マージツール「etmerge」

マージツールは Windows の Interschema が提供するユティリティプログラムです。この節では，マージツールはどのような場合に使用するのか，又その使用方法について説明します。

7.1.1 マージツールとは

マージツールは，FDL ファイル又は DTD ファイルの変更内容を MDL ファイルにマージできます。

マージツールには四つの機能があります。

次に，各機能について説明します。

(1) マージ機能とは

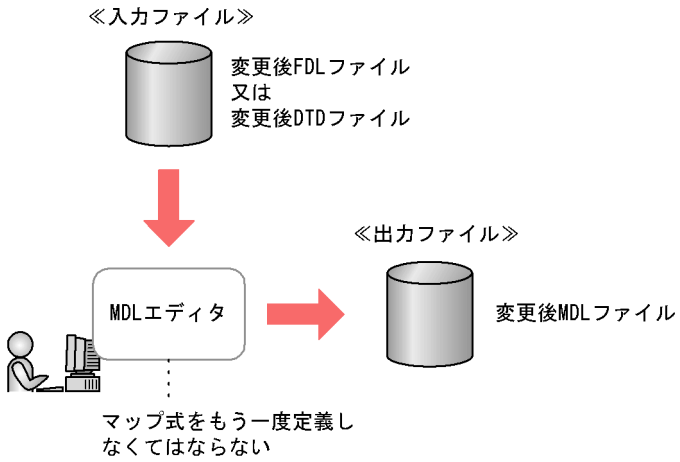
MDL ファイル中の既存のマップ式を保持しながら，FDL ファイル又は DTD ファイルの変更内容を MDL ファイルにマージする機能です。マージ機能を使用することで，再マッピングの作業を最小限に抑えることができます。

マージ機能は，MDL エディタの［編集］－［マージ］コマンドでも実行できます。

MDL ファイルを作成した後に要素のサイズや出現回数などを変更する場合は，FDL ファイル又は DTD ファイルでフォーマットを変更してから，MDL ファイル内で変更前のフォーマットを削除して，変更後のフォーマットを読み込んで MDL ファイルを作成し直す必要があります。

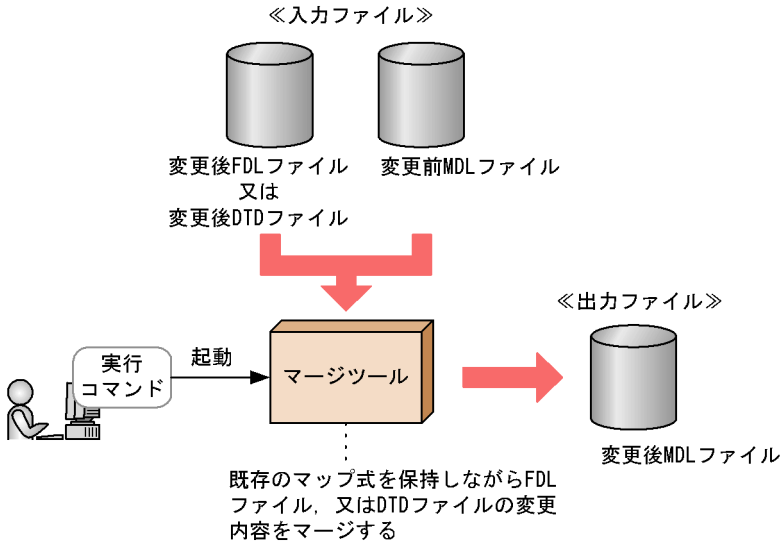
入力側のフォーマットを変更した場合は，変更前のフォーマットを削除して変更後のフォーマットを読み込んでも，既存のマップ式は保持されるため，コンポーネント名やツリー構造を変更していなければそのまま使用できます。しかし，出力側のフォーマットを変更した場合は，変更前のフォーマットを削除すると既存のマップ式も削除されてしまうため，変更後のフォーマットを読み込んで再マッピングする作業が発生します。

図 7-1 マージ機能を使用しない場合



このような場合にマージ機能を使用すると，名前やツリー構造を変更していないコンポーネントの既存のマップ式を保持しながら，FDL ファイル又は DTD ファイルの変更内容を MDL ファイルにマージできます。この結果，再マッピングする必要があるのは，コンポーネント名やツリー構造を変更した箇所だけになり，再マッピングの作業を最小限に抑えることができます。

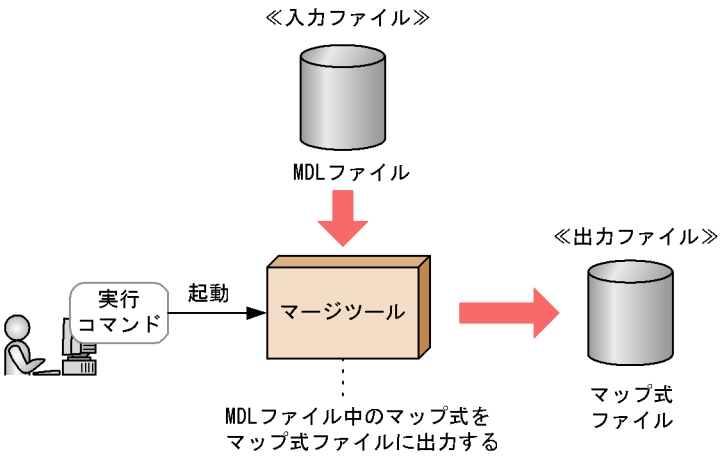
図 7-2 マージ機能を使用した場合



(2) マップ式出力機能とは

MDL ファイル中のマップ式をテキスト形式で記述したファイル「マップ式ファイル」に出力する機能です。

マップ式出力機能は，MDL エディタの［ファイル］－［外部出力］コマンドでも実行できます。



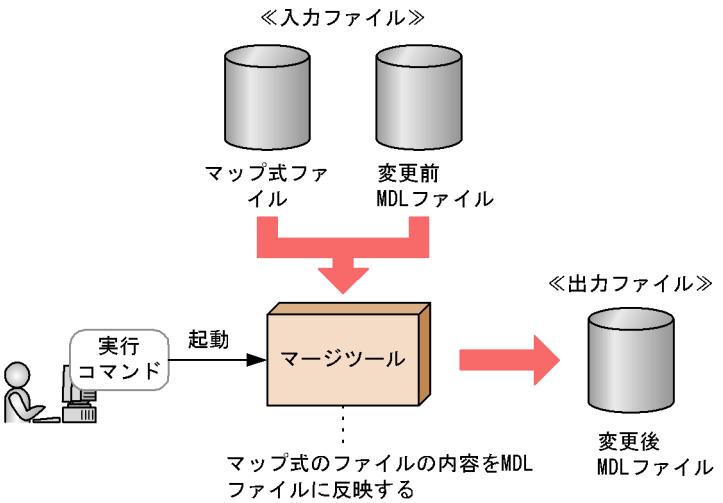
マップ式ファイルには，任意のコンポーネント以下のマップ式を出力できます。マップ式ファイルの形式については，「付録 E.1　マップ式ファイル」を参照してください。

出力したマップ式は，マップ式ファイル上で直接編集できます。マップ式ファイルの編集内容は，マッピング機能を使用して MDL ファイルに反映できます。

(3) マッピング機能とは

マップ式ファイルの内容を MDL ファイルに反映する機能です。マップ式ファイルの編集内容を MDL ファイルに反映できます。

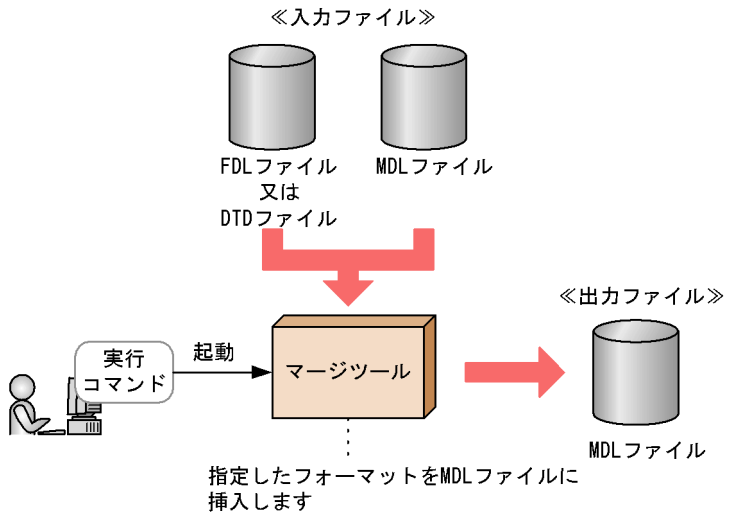
マッピング機能は，MDL エディタの［編集］－［マージ］コマンドでも実行できます。



(4) フォーマット挿入機能とは

指定したFDLフォーマット，又は指定したDTDフォーマットをMDLファイルに挿入する機能です。

フォーマット挿入機能は，MDLエディタの［挿入］－［フォーマット］コマンド又は［挿入］－［DTDフォーマット］でも実行できます。



以上の四つの機能を実行するには，コマンド「etmerge」でマージツールを起動して，実行したい機能の引数を指定します。

次に，マージツールのコマンド「etmerge」について機能ごとに説明します。コマンド「etmerge」は，次のディレクトリ下に格納されています。

Interschemaのインストールディレクトリ¥bin

7.1.2 FDLマージ機能（/IFオプション）

マージ機能を実行して，FDLファイルの変更内容をMDLファイルにマージするには，コマンド「etmerge」の引数として「/IF」オプションを指定します。

(1) 形式

```
etmerge  /IF  旧MDLファイル  新MDLファイル  FDLファイル  フォーマット名
      [マップ式ファイル]
```

(2) 引数

/IF

MDLファイルの既存のマップ式，変数式，コンポーネント未使用情報，グループ単

位出力指定情報，MDL 上で編集された式，及び順序決定値を保持しながら，FDL ファイルの変更内容を MDL ファイルにマージする機能を実行するオプションです。

旧 MDL ファイル
変更前の MDL ファイル名を指定します。種別は入力です。

新 MDL ファイル
FDL ファイルをマージした後の MDL ファイル名を指定します。旧 MDL ファイルと同じファイルを指定してもかまいません。種別は出力です。

FDL ファイル
MDL ファイルにマージする FDL ファイル名を指定します。種別は入力です。

フォーマット名
MDL ファイルにマージする FDL ファイル中のフォーマット名を指定します。種別は入力です。

マップ式ファイル
旧 MDL ファイルに定義されているすべてのマップ式を出力するマップ式ファイル名を指定します。種別は出力です。この指定は省略できます。省略した場合は，「MAPEXP.txt」ファイル（*Interschema のインストールディレクトリ*¥bin 下に格納）にマップ式を出力します。

(3) /IF オプション指定時の注意事項

- マップ式などの保持は，コンポーネントのグローバル名をキーにして行います。したがって，グローバル名が変更された場合は，そのコンポーネントに定義されていたマップ式は保持されません。また，コンポーネントの削除や名前の変更があっても，そのコンポーネントを参照する式の内容は更新されないで元の文字列のまま引き継がれます。
- 保持した式で記述されているコンポーネントのパス名の正当性は，検証されません。FDL ファイルのツリー構造やコンポーネント名を変更した場合などは，式を確認して再マッピングしてください。
- マップ式ファイルに記述されているコンポーネントのうち，MDL ファイル中になかったコンポーネントは，該当するテキスト部分がマージログファイル「MERGELOG.txt」（*Interschema のインストールディレクトリ*¥bin 下に格納）に出力されます。MDL ファイルにマージするコンポーネントがある場合で，そのコンポーネントの変数式で定義した変数が，マージ後のフォーマットにないときは，コンポーネントのグローバル名及び変数式がマージログファイルに出力されます。
- FDL ファイル中のグループ単位出力指定に変更があっても，その変更内容は MDL ファイルにマージされません（変更前の指定内容を保持します）。このような場合は，MDL エディタで MDL ファイル中のグループ単位出力指定の内容を変更してから，マージツールを実行してください。
- 変数式は，コンポーネント単位で扱われます。そのため，マージ元に複数の変数の変数式が定義されていても，マージ後のフォーマットで一つの変数の変数式だけ定義し

た場合は，定義しなかった変数に対する変数式は消えてしまいます。また，マージ後のフォーマットに存在しない変数に対する変数式は，設定されません。

- MDL エディタ上で直接編集した，又はマージツールの /IT オプションで設定した，順序決定式，順序決定値，デフォルト式，条件式，出現回数決定式，及びサイズ決定式の内容は，マージするコンポーネントがマージ先のフォーマットにある場合に保持されます。ただし，順序決定式を定義したコンポーネントが逐次構造になる場合など，コンポーネントの種別又は状態が，マージ先で式や値を定義できない状態に変換される場合は保持されません。

（4） 実行例

FDL マージ機能の実行例を次に示します。

FDL ファイル「UPDATE.fdl」中のフォーマット「Format1」の内容を，旧 MDL ファイル「OLD.mdl」にマージして，新 MDL ファイル「NEW.mdl」に出力します。旧 MDL ファイル「OLD.mdl」に定義されていたマップ式は，マップ式ファイル「MAPFILE.txt」に出力します。

```
etmerge  /IF  OLD.mdl  NEW.mdl  UPDATE.fdl  Format1  MAPFILE.txt
```

7.1.3 DTD マージ機能（/ID オプション）

マージ機能を実行して，DTD ファイルの変更内容を MDL ファイルにマージするには，コマンド「etmerge」の引数として「/ID」オプションを指定します。

（1） 形式

```
etmerge  /ID  旧MDLファイル  新MDLファイル  DTDファイル  フォーマット名
      [マップ式ファイル]
```

（2） 引数

/ID

MDL ファイルの既存のマップ式，コンポーネント未使用情報，グループ単位出力指定情報，MDL 上で編集された式及び順序決定値を保持しながら，DTD ファイルの変更内容を MDL ファイルにマージする機能を実行するオプションです。

旧 MDL ファイル

変更前の MDL ファイル名を指定します。種別は入力です。

新 MDL ファイル

DTD ファイルをマージした後の MDL ファイル名を指定します。旧 MDL ファイルと同じファイルを指定してもかまいません。種別は出力です。

DTD ファイル

MDL ファイルにマージする DTD ファイル名を指定します。種別は入力です。

フォーマット名

MDL ファイルにマージする DTD フォーマット名を指定します。種別は入力です。

マップ式ファイル

旧 MDL ファイルに定義されているすべてのマップ式を出力するマップ式ファイル名を指定します。種別は出力です。この指定は省略できます。省略した場合は，「MAPEXP.txt」ファイル（*Interschema のインストールディレクトリ*¥bin 下に格納）にマップ式を出力します。

(3) /ID オプション指定時の注意事項

- マップ式などの保持は，コンポーネントのグローバル名をキーにして行います。したがって，グローバル名が変更された場合は，そのコンポーネントに定義されていたマップ式は保持されません。また，コンポーネントの削除や名前の変更があっても，そのコンポーネントを参照する式の内容は更新されないで元の文字列のまま引き継がれます。
- 保持した式で記述されているコンポーネントのパス名の正当性は，検証されません。DTD ファイルのツリー構造やコンポーネント名を変更した場合などは，式を確認して再マッピングしてください。
- マップ式ファイルに記述されているコンポーネントのうち，MDL ファイル中になかったコンポーネントは，該当するテキスト部分がマージログファイル「MERGELOG.txt」（*Interschema のインストールディレクトリ*¥bin 下に格納）に出力されます。MDL ファイル中にマージするコンポーネントがある場合で，そのコンポーネントの変数式で定義した変数がマージ後のフォーマットにないときは，コンポーネントのグローバル名及び変数式がマージログファイルに出力されます。
- 変数式は，マージ先に対応する変数がある場合だけ保持されるため，DTD マージでは結果として削除されます。
- マージする DTD フォーマットが再帰構造を含む場合，マージ前の展開ネスト数で再帰構造が展開されます。
  マージ前に再帰構造がなかった場合の展開ネスト数は 1 となります。
  マージ後の DTD フォーマットのルートとする要素の候補が複数存在する場合，マージ前と同じ名前の要素があれば，これがルートとして選択されます。同じ名前の要素がない場合は，自動的にルートとする要素が選択されます。マージ後のルートとする要素の候補が一つだけである場合は，マージ前と異なる名前であってもルートとして選択されます。
- 対象となる DTD フォーマットが再帰構造を含む場合，再帰構造を展開するときの最大コンポーネント数は 100,000 として処理されます。コンポーネント数が 100,000 を超える場合は，MDL エディタを使用してください。
- MDL エディタ上で直接編集した，又はマージツールの /IT オプションで設定した，順序決定式，順序決定値，デフォルト式，条件式，出現回数決定式，及びサイズ決定式の内容は，マージするコンポーネントがマージ先のフォーマットにある場合に保持されます。ただし，順序決定式を定義したコンポーネントが逐次構造になる場合など，

コンポーネントの種別又は状態が，マージ先で式や値を定義できない状態に変換される場合は保持されません。

(4) 実行例

DTD マージ機能の実行例を次に示します。

旧 MDL ファイル「OLD.mdl」の DTD フォーマット「Format1」の内容を，DTD ファイル「UPDATE.dtd」の内容にマージして，新 MDL ファイル「NEW.mdl」に出力します。旧 MDL ファイル「OLD.mdl」に定義されていたマップ式は，マップ式ファイル「MAPFILE.txt」に出力します。

```
etmerge  /ID  OLD.mdl  NEW.mdl  UPDATE.dtd  Format1  MAPFILE.txt
```

7.1.4 マップ式出力機能（/OT オプション）

マップ式出力機能を実行するには，コマンド「etmerge」の引数として「/OT」オプションを指定します。

(1) 形式

```
etmerge  /OT  MDLファイル  マップ式ファイル  [コンポーネント名]
```

(2) 引数

/OT

MDL ファイル中のマップ式，変数式，コンポーネント未使用情報，グループ単位出力指定情報，MDL 上で編集された式及び順序決定値をマップ式ファイルに出力する機能を実行するオプションです。任意のコンポーネント以下のマップ式を出力できます。

MDL ファイル

マップ式ファイルを出力する MDL ファイル名を指定します。種別は入力です。

マップ式ファイル

MDL ファイル中のマップ式を出力するマップ式ファイル名を指定します。種別は出力です。

コンポーネント名

マップ式ファイルにマップ式を出力するコンポーネント名を指定します。ただし，グローバル名にはインデクスを含めないで指定します。子を持つコンポーネントを指定した場合は，その子孫コンポーネントのマップ式も出力対象となります。この指定は省略できます。省略した場合は，MDL ファイル中にあるすべてのコンポーネントのマップ式が出力対象になります。種別は入力です。

(3) 実行例

マップ式出力機能の実行例を次に示します。

マップ式ファイル「MAPFILE.txt」に，MDL ファイル「REFER.mdl」中のコンポーネント「Format1@Struct1」以下のすべてのコンポーネントのマップ式を出力します。

```
etmerge  /OT  REFER.mdl  MAPFILE.txt  Format1@Struct1
```

7.1.5 マッピング機能（/IT オプション）

マッピング機能を実行するには，コマンド「etmerge」の引数として「/IT」オプションを指定します。

(1) 形式

```
etmerge  /IT  旧MDLファイル  新MDLファイル  マップ式ファイル
```

(2) 引数

/IT

マップ式ファイルの内容を MDL ファイルに反映する機能を実行するオプションです。

旧 MDL ファイル

変更前の MDL ファイル名を指定します。種別は入力です。

新 MDL ファイル

マップ式ファイルの内容を反映した後の MDL ファイル名を指定します。旧 MDL ファイルと同じファイルを指定してもかまいません。種別は出力です。

マップ式ファイル

MDL ファイルに反映するマップ式を記述したマップ式ファイル名を指定します。種別は入力です。

(3) /IT オプション指定時の注意事項

- マップ式ファイルに記述されているコンポーネントのうち，MDL ファイル中になかったコンポーネントについては，該当するテキスト部分がマージログファイル「MERGELOG.txt」（*Interschema のインストールディレクトリ*¥bin 下に格納）に出力されます。MDL ファイルにマージするコンポーネントがある場合で，そのコンポーネントの変数式で定義した変数がマージ後のフォーマットにないときは，コンポーネントのグローバル名及び変数式がマージログファイルに出力されます。
- マッピングしたマップ式などで記述されているコンポーネントのパス名は，正当性をチェックしません。ツリー構造又はコンポーネント名を変更した場合は，式を確認して再定義してください。

- 変数式は，コンポーネント単位で扱われます。そのため，マージ元に複数の変数の変数式が定義されていても，マージ後のマップ式ファイルで一つの変数の変数式だけ定義した場合は，定義しなかった変数に対する変数式は消えてしまいます。また，マージ後のフォーマットに存在しない変数に対する変数式は，設定されません。
- マップ式ファイルに記述された順序決定式，順序決定値，デフォルト式，条件式，出現回数決定式，及びサイズ決定式の内容は，マッピング対象のコンポーネントがマッピング先のフォーマットにある場合に設定されます。ただし，順序決定式を定義した構造のコンポーネントが逐次構造である場合など，式又は値がマッピングできないときは無視されます。

(4) 実行例

マッピング機能の実行例を次に示します。

マップ式ファイル「MAPFILE.txt」の内容を旧 MDL ファイル「OLD.mdl」に反映して，新 MDL ファイル「NEW.mdl」として出力します。

```
etmerge  /IT  OLD.mdl  NEW.mdl  MAPFILE.txt
```

7.1.6 FDL フォーマット挿入機能（/AF オプション）

フォーマット挿入機能を実行して，指定した FDL フォーマットを MDL ファイルに挿入するには，コマンド「etmerge」の引数として「/AF」オプションを指定します。

(1) 形式

```
etmerge  /AF  旧MDLファイル  新MDLファイル  FDLファイル
                    [{/IN フォーマット名｜/OUT フォーマット名}... ]
```

(2) 引数

/AF

指定した FDL フォーマットを MDL ファイルに挿入する機能を実行するオプションです。

旧 MDL ファイル

変更前の MDL ファイル名を指定します。種別は入力です。

新 MDL ファイル

FDL ファイルを挿入した後の MDL ファイル名を指定します。旧 MDL ファイルと同じファイルを指定してもかまいません。種別は出力です。

FDL ファイル

MDL ファイルに挿入する FDL ファイル名を指定します。種別は入力です。

フォーマット名

MDL ファイルに挿入する FDL ファイルのフォーマット名を指定します。入力フォーマットとして挿入する場合は「/IN」，出力フォーマットとして挿入する場合は「/OUT」をそれぞれフォーマット名の前に指定します。種別は入力です。

(3) /AF オプション指定時の注意事項

- 旧 MDL ファイルがない場合は，新 MDL ファイルを新規に作成します。
- フォーマット名を省略した場合は，FDL ファイル内のフォーマットの中から，入出力兼用フォーマット以外で検証済みのフォーマットすべてを挿入します。

(4) 実行例

FDL フォーマット挿入機能の実行例を次に示します。

「OLD.mdl」に，「ADD.fdl」のフォーマット「Format1」を入力側に，フォーマット「Format2」を出力側に挿入した結果を「NEW.mdl」として出力します。

```
etmerge  /AF  OLD.mdl  NEW.mdl  ADD.fdl  /IN  Format1 /OUT  Format2
```

7.1.7 DTD フォーマット挿入機能（/AD オプション）

フォーマット挿入機能を実行して，指定した DTD フォーマットを MDL ファイルに挿入するには，コマンド「etmerge」の引数として「/AD」オプションを指定します。

(1) 形式

```
etmerge  /AD  旧MDLファイル  新MDLファイル  DTDファイル
                    {/IN フォーマット名 | /OUT フォーマット名}
                    [ルート要素名[展開ネスト数]]
```

(2) 引数

/AD

指定した DTD フォーマットを MDL ファイルに挿入する機能を実行するオプションです。

旧 MDL ファイル

変更前の MDL ファイル名を指定します。種別は入力です。

新 MDL ファイル

DTD ファイルを挿入した後の MDL ファイル名を指定します。旧 MDL ファイルと同じファイルを指定してもかまいません。種別は出力です。

DTD ファイル

MDL ファイルに挿入する DTD ファイル名を指定します。種別は入力です。

フォーマット名

MDL ファイルに挿入する DTD フォーマット名を 256 バイト以内で指定します。入力フォーマットとして挿入する場合は「/IN」，出力フォーマットとして挿入する場合は「/OUT」をそれぞれフォーマット名の前に指定します。種別は入力です。

ルート要素名

DTD フォーマットのルートとする要素名を指定します。指定しない場合，自動的にルートとする要素が選択されます。挿入する DTD フォーマットのルートとする要素の候補が一つだけである場合は，指定しても意味がありません。種別は入力です。

展開ネスト数

DTD フォーマットの再帰構造の展開ネスト数を 0 ～ 250 の範囲で指定します。ルート構造が再帰構造の場合は，1 ～ 250 の範囲で指定します。指定しない場合，自動的に 1 となります。挿入する DTD フォーマットに再帰構造がない場合は，指定しても意味がありません。種別は入力です。

（3） /AD オプション指定時の注意事項

- 旧 MDL ファイルがない場合は，新 MDL ファイルを新規に作成します。
- 対象となる DTD フォーマットが再帰構造を含む場合，再帰構造を展開するときの最大コンポーネント数は 100,000 として処理されます。コンポーネント数が 100,000 を超える場合は，MDL エディタを使用してください。

（4） 実行例

DTD フォーマット挿入機能の実行例を次に示します。

「OLD.mdl」に，「ADD.dtd」の内容を DTD フォーマット「Format1」として出力側に挿入した結果を「NEW.mdl」として出力します。DTD フォーマットのルート要素は「ROOT1」と，再帰構造の展開ネスト数は「3」とします

```
etmerge  /AD  OLD.mdl  NEW.mdl  ADD.dtd  /OUT  Format1  ROOT1  3
```

7.1.8 注意事項

マージツール実行時の注意事項を次に示します。

● トランスレータ実行中にはマージツールを起動しないでください。

● MDL ファイルの検証
マージツールで出力した MDL ファイルは検証されていない状態になります。MDL ファイルを使用してデータを変換する前に，MDL エディタでファイルを検証してください。

● エラー発生時の処理
エラーが発生した場合は，標準出力にエラーの内容（戻り値，エラー内容の説明など）

を出力します。エラー内容については,「7.1.9　戻り値」を参照してください。

7.1.9　戻り値

マージツールの戻り値の値と意味を次に示します。

表 7-1　戻り値

値	エラーレベル	意味
0	正常	正常終了しました。
1	エラー	オプション又は引数の指定が不正です。
2	エラー	ファイルがないか，読み取り権限がありません。エラーの発生したファイル名が表示されます。
3	エラー	ファイル又はディレクトリに書き込み権限がありません。エラーの発生したファイル名が表示されます。
4	エラー	メモリ不足です。
5	エラー	ファイル IO エラーです。ファイル名が表示されます。
100	エラー	FDL ファイル又は MDL ファイルのバージョンが不正です。エラーの発生したファイル名が表示されます。
101	エラー	指定したフォーマット名が，FDL ファイル又は MDL ファイル中にありません。エラーの発生した FDL ファイル名，MDL ファイル名，及びフォーマット名が表示されます。
102	エラー	指定した FDL フォーマットが検証されていません。エラーの発生した FDL ファイル名及びフォーマット名が表示されません。
103	エラー	MDL ファイルへのフォーマット挿入時に検証エラーが発生しました。
104	エラー	挿入できるフォーマットがない。エラーの発生した FDL ファイル名が表示されます。
105	エラー	同一名称のフォーマットが挿入先の MDL ファイルにある。FDL ファイル名及びフォーマット名が表示されます。
106	エラー	挿入できない DTD フォーマットが指定された。又は DTD フォーマット挿入でエラーが発生しました。
107	エラー	指定されたフォーマット名が不正です。
108	エラー	FDL ファイル又は MDL ファイルの内容が不正です。不正な内容を含む FDL ファイル又は MDL ファイルのファイル名が表示されます。
109	エラー	再帰構造を展開したコンポーネントの最大数が制限値を超えました。
201	エラー	マップ式ファイルの記述が不正です。エラーの発生したマップ式ファイル名及びエラー位置が表示されます。
301	エラー	指定したコンポーネントが MDL 中にありません。エラーの発生した MDL ファイル名及びコンポーネント名が表示されます。
801	ワーニング	一部のコンポーネントの情報がマージできませんでした。

値	エラーレベル	意味
802	ワーニング	一部の変数式の情報がマージできませんでした。※
901	エラー	内部エラーです。保守員に連絡してください。

注※

マージできないコンポーネント及びマージできない変数式がある場合は，戻り値「801」が出力されます。

7.1.10 ヘルプの表示

マージツールのヘルプを表示するには，コマンド「etmerge」の引数として「/HELP」又は「/?」オプションを指定します。指定例を次に示します。

```
etmerge /HELP
```

8 インタフェース

この章では，Interschema が提供するインタフェースの概要及び注意事項について説明します。

8.1 インタフェースの概要

この節では，Interschema が提供するインタフェースの概要を説明します。
Interschema が提供するインタフェースには，データ変換処理 API 及びユーザ組み込み関数の出口関数があります。

8.1.1 データ変換処理 API の概要

データ変換処理 API は，C 言語の実行環境又は Java 言語の実行環境で，ユーザ業務プログラムからトランスレータを起動して，データ変換を行うために必要なインタフェースです。Interschema は，次の二つのデータ変換処理 API を提供します。

- C 言語の実行環境で使用するデータ変換処理 API（以降，データ変換処理 API（C 言語）と呼びます）
- Java 言語の実行環境で使用するデータ変換処理 API（以降，データ変換処理 API（Java 言語）と呼びます）

データ変換処理 API を使用すると，既存のユーザ業務プログラムにトランスレータの機能を容易に組み込めます。また，ファイル化されたデータだけでなく，メモリ上のデータも変換できます。

データ変換処理 API（C 言語）の詳細については，「9. データ変換処理 API（C 言語）」を参照してください。データ変換処理 API（Java 言語）の詳細については，「10. データ変換処理 API（Java 言語）」を参照してください。

8.1.2 ユーザ組み込み関数の出口関数の概要

FDL ファイル又は MDL ファイルに変換情報を定義するとき，Interschema が標準で提供している関数以外に，ユーザが独自に定義したユーザ組み込み関数を使用できます。Interschema は，データ変換を実行中に，MDL ファイルに定義されたユーザ組み込み関数から，ユーザが C 言語又は Java 言語で作成したユーザプログラムを出口関数として呼び出すインタフェースを提供します。

ユーザ組み込み関数については，「11. ユーザ組み込み関数」を参照してください。C 言語の出口関数の詳細については，「11.4 C 言語の出口関数の作成」を参照してください。Java 言語の出口関数の詳細については，「11.5 Java 言語の出口関数の作成」を参照してください。

8.2 C 言語及び Java 言語のインタフェースを混在させて使用する場合の注意事項

この節では，Interschema が提供する C 言語のインタフェース，及び Java 言語のインタフェースを混在させて使用する場合の注意事項について説明します。

- 同一プロセス内で，データ変換処理 API（C 言語）及びデータ変換処理 API（Java 言語）を混在させて使用しないでください。
- ettrans コマンドは C 言語になります。
- データ変換 API（C 言語）及びデータ変換 API（Java 言語），並びに C 言語の出口関数及び Java 言語の出口関数を混在させて使用する場合の制限事項を，次の表に示します。

表 8-1 異なる言語のインタフェースを混在させて使用する場合の制限事項

条件		結果		備考
データ変換 API の対応言語	出口関数の対応言語	出口関数を呼び出す	データ変換 API から出口関数にオブジェクトを渡す	
C 言語	C 言語	○	○	システム情報ファイル「ettrans.ini」で USE02IF が指定された出口関数にだけ，オブジェクトを渡します。
C 言語	Java 言語	×	×	データ変換 API の呼び出し時に，出口関数が不正なため変換処理を終了します（メッセージ KBET0030T-E を出力します）。
Java 言語	C 言語	○	×	システム情報ファイル「ettrans.ini」で USE02IF が指定された出口関数には，オブジェクトとして NULL を渡します。
Java 言語	Java 言語	○	○	－

（凡例）

○：できます。

×：できません。

－：特にありません。

9 データ変換処理 API（C 言語）

この章では，データ変換処理 API（C 言語）について説明します。

9.1 データ変換処理 API（C 言語）の概要

Interschema は，C 言語の実行環境で，ユーザ業務プログラムからトランスレータを起動してデータ変換を行うために必要な，データ変換処理 API（C 言語）を提供します。

この節では，データ変換処理 API（C 言語）の機能概要，実行順序，及び実行時の注意事項について説明します。データ変換処理 API（C 言語）の関数を使用する前に，必ずお読みください。

9.1.1 データ変換処理 API（C 言語）の機能概要

データ変換処理 API（C 言語）の関数一覧を次に示します。

表 9-1 データ変換処理 API（C 言語）の関数一覧

項番	関数名	機能概要	種別
1	ETtrans2Init	トランスレータの初期化	トランスレータ操作関数
2	ETtrans2End	トランスレータの終了処理	
3	ETtrans2CreateMdlInfo	MDL 情報の生成	MDL 情報操作関数
4	ETtrans2ReleaseMdlInfo	MDL 情報の解放	
5	ETtrans2UpdateMdlInfo	MDL 情報の更新	
6	ETtrans2CreateThreadContext	スレッド固有情報の生成	スレッド固有情報操作関数
7	ETtrans2ReleaseThreadContext	スレッド固有情報の解放	
8	ETtrans2UpdateThreadContext	スレッド固有情報の更新	
9	ETtrans2Exec	変換実行	変換実行関数
10	ETtrans2GetDlProperty	DL プロパティ情報の取得	DL プロパティ操作関数
11	ETtrans2ReleaseDlProperty	DL プロパティ情報の解放	

データ変換処理 API（C 言語）の各関数については，「9.2 データ変換処理 API（C 言語）の関数仕様」を参照してください。

9.1.2 データ変換処理 API（C 言語）の実行順序

データ変換処理 API（C 言語）の関数の実行順序は，処理の方法によって次の三つに分けられます。

- シングルスレッドの場合
- マルチスレッド（スレッド内で MDL 情報を生成しない）の場合

- マルチスレッド（スレッド内で MDL 情報を生成する）の場合

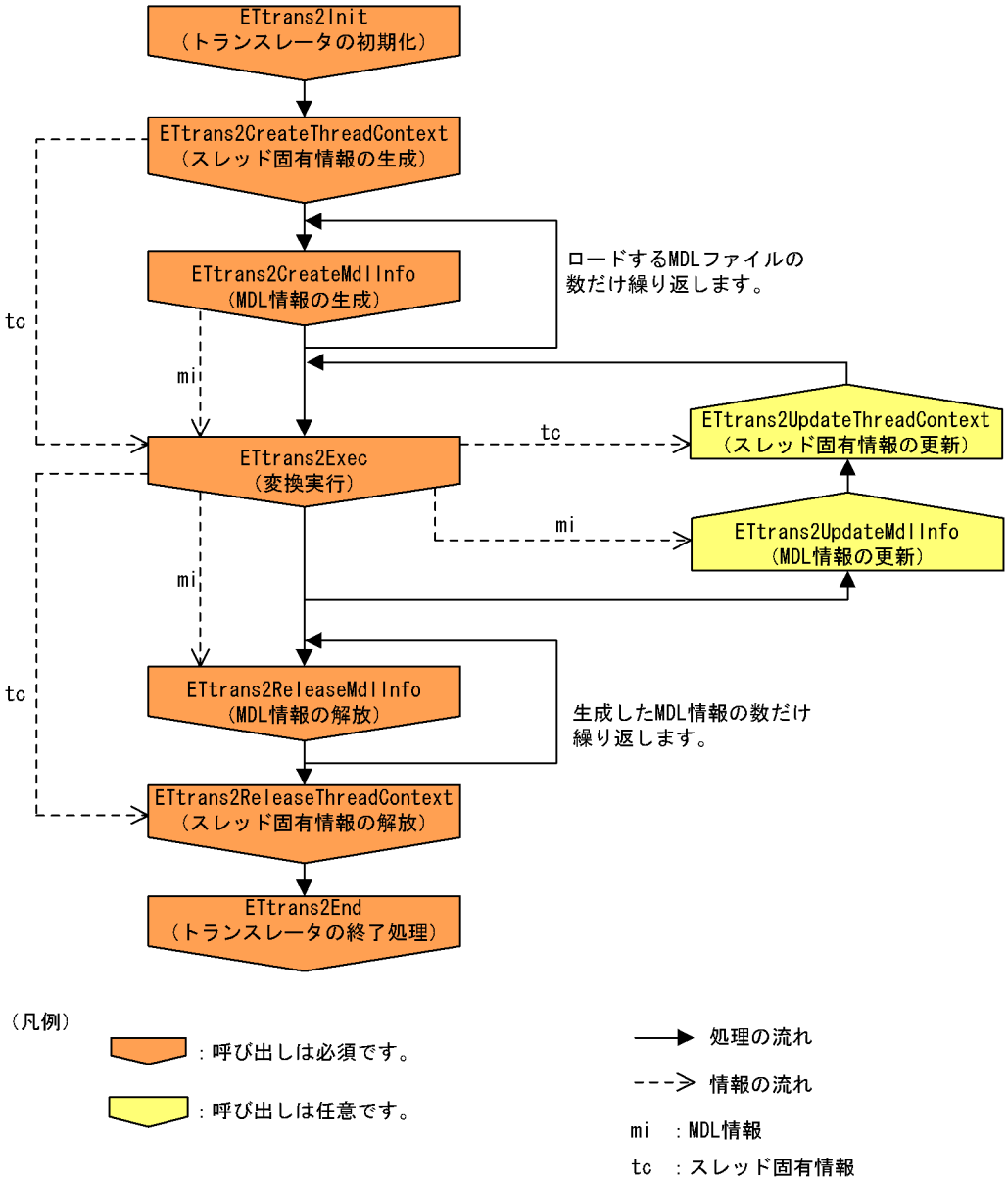
各処理での関数の実行順序を次に示します。関数を実行する場合は，必ずこの順序で行ってください。

ETtrans2GetDlProperty 関数，及び ETtrans2ReleaseDlProperty 関数は，ETtrans2Init 関数から ETtrans2End 関数の間の任意の箇所で呼び出せます。ただし，ETtrans2GetDlProperty 関数，ETtrans2ReleaseDlProperty 関数の順で呼び出してください。

（1）シングルスレッドの場合

シングルスレッドでの関数の実行順序を次に示します。

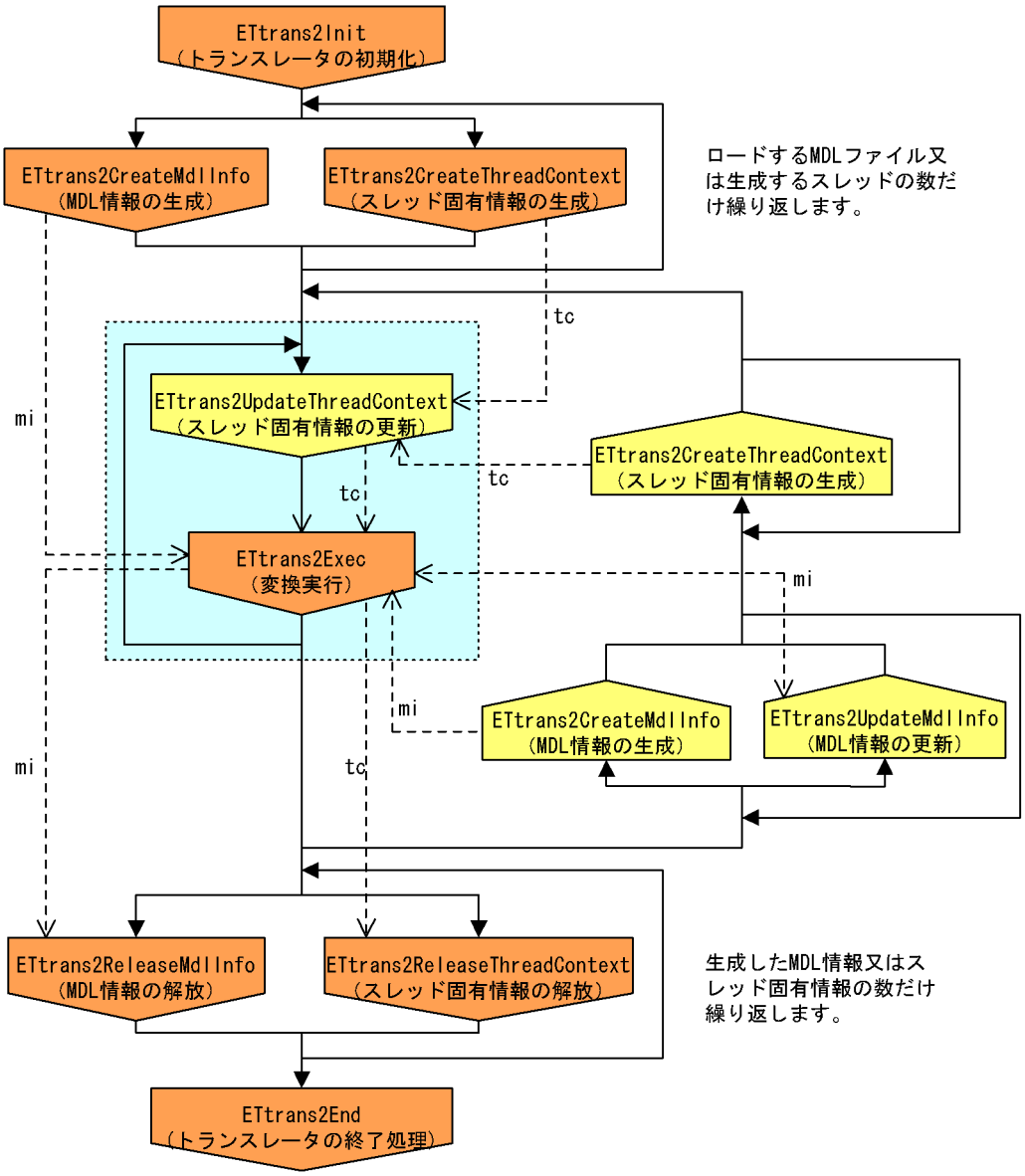
図 9-1 シングルスレッドでの関数の実行順序



(2) マルチスレッド（スレッド内で MDL 情報を生成しない）の場合

マルチスレッド（スレッド内で MDL 情報を生成しない）での関数の実行順序を次に示します。

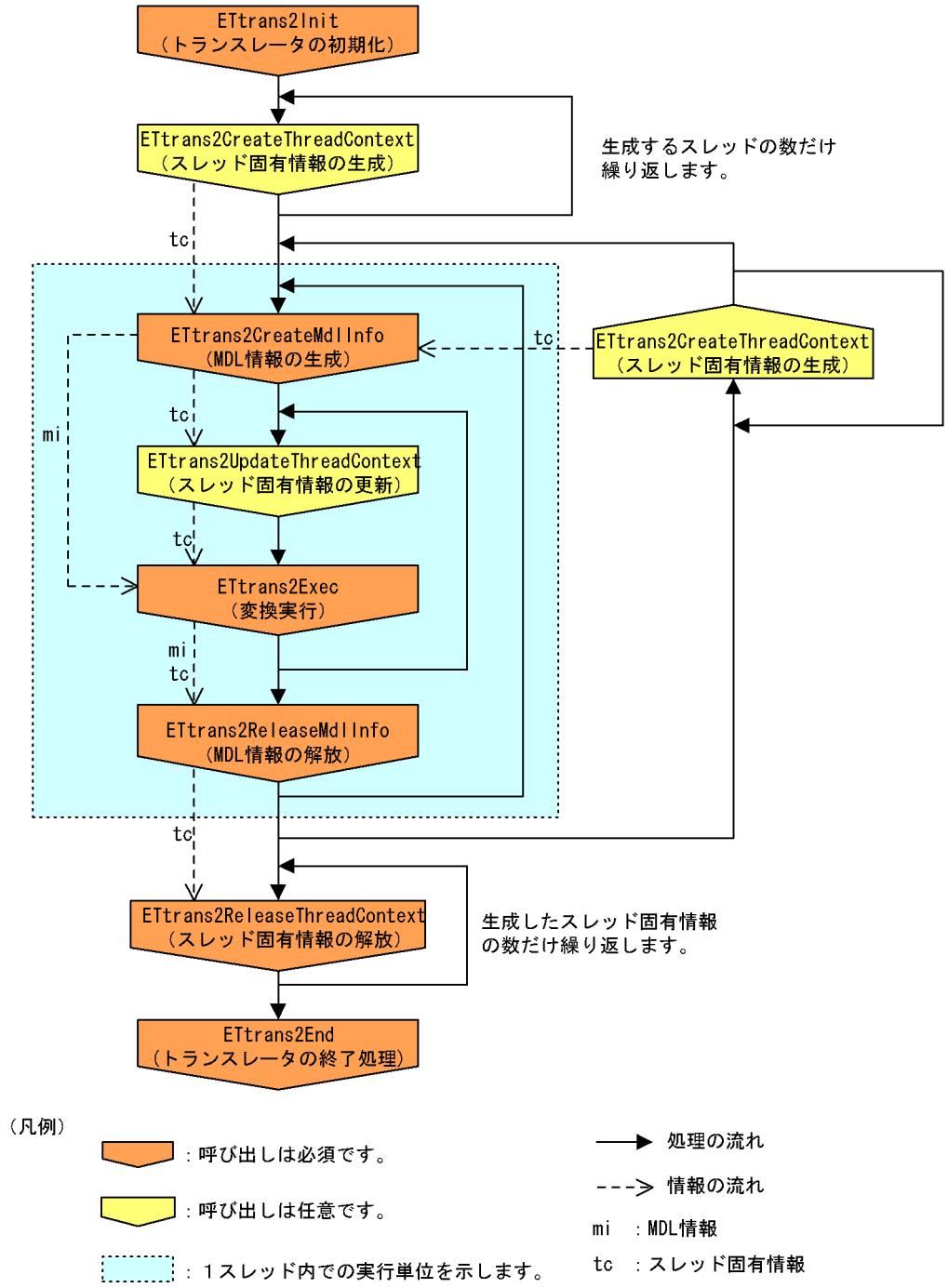
図 9-2 マルチスレッド（スレッド内で MDL 情報を生成しない）での関数の実行順序



（3）マルチスレッド（スレッド内で MDL 情報を生成する）の場合

マルチスレッド（スレッド内で MDL 情報を生成する）での関数の実行順序を次に示します。

図 9-3　マルチスレッド（スレッド内で MDL 情報を生成する）での関数の実行順序



9.1.3 データ変換処理 API（C 言語）実行時の注意事項

データ変換処理 API（C 言語）の関数を使用する場合は，次の点に注意してください。

- データ変換処理 API（C 言語）は「9.1.2　データ変換処理 API（C 言語）の実行順序」に従って使用してください。異なる遷移で実行した場合，不正な結果となります。
- スレッド固有情報は変換を実行するスレッドごとに生成してください。各スレッドには，異なるスレッド固有情報を渡してください。
- 一つの処理の中で，データ変換処理 API（C 言語）と互換 API を混在して使うことはできません。互換 API については，「付録 D　Interschema バージョン 1 との互換 API」を参照してください。
- マルチスレッド（スレッド内で MDL 情報を生成しない）の場合，変更する MDL 情報がほかのスレッドで使用中のときは，ETtrans2UpdateMdlInfo 関数を呼び出さないでください。
- マルチスレッド（スレッド内で MDL 情報を生成しない）の場合とマルチスレッド（スレッド内で MDL 情報を生成する）の場合を混在させて処理する場合で，MDL 情報をスレッド内で生成するときは，MDL 情報はそのスレッド内だけで使用してください。
- マルチスレッドの場合で，動作中のスレッドがあるときは，呼び出しが任意の関数（ETtrans2UpdateThreadContext 関数は除きます）を呼び出さないでください。

9.2 データ変換処理 API（C 言語）の関数仕様

この節では，データ変換処理 API（C 言語）の各関数の仕様について説明します。データ変換処理 API（C 言語）の関数の記述形式を次に示します。

形式

関数の記述形式を示します。

引数

関数の引数及び入出力の種別を示します。

説明

関数の機能について説明します。

戻り値

関数の戻り値を示します。

次にデータ変換処理 API（C 言語）の各関数の仕様を示します。

9.2.1 ETtrans2Init（トランスレータの初期化）

形式

```
#include  "ETtrans.h"

int ETtrans2Init (const char *cszLogFile)
```

引数

引数	種別	内容
cszLogFile	入力	ログファイル名を指定します。

説明

トランスレータを初期化します。この関数を呼び出すのは，一つのプロセスの中で１回だけです。

この関数を呼び出したら，ETtrans2End 関数が呼び出されるまで，スレッド上のすべての関数処理のログ情報は，引数 cszLogFile で指定したファイルに出力されます。

引数 cszLogFile に指定したファイルを作成又は更新できなかった場合，若しくは引数 cszLogFile に NULL を指定した場合は，エラー情報はデフォルトのログファイル「errlog.txt」に出力されます。デフォルトのログファイルを作成又は更新できなかった場合は，標準出力に出力されます。

この関数内でログファイルに情報を出力する場合，ログファイルの最大サイズは初期値（1MB）となります。ただし，システム情報ファイル「ettrans.ini」でログを

出力しないように指定している場合は，引数 cszLogFile 指定に関係なくログは出力されません。

戻り値

戻り値	内容
0x00000000	正常に終了しました。
0x01000010※	指定されたログファイルがオープンできないため，ログを初期値のログファイル名で出力，又は標準出力に出力しました。
0x01000040※	出口関数の定義に不正があります。
0x01000100※	「ettrans.ini」の定義に不正があります。
0x04000002	メモリ不足です。

注※

0x01000010，0x01000040，及び 0x01000100 は，ほかの値と同時に設定される場合があります。

9.2.2 ETtrans2End（トランスレータの終了処理）

形式

```
#include  "ETtrans.h"

int ETtrans2End()
```

引数

なし

説明

トランスレータを終了します。この関数を呼び出すのは，一つのプロセスの中で 1 回だけです。

データ変換処理を終了するには，この関数を実行して正常に終了したことを確認する必要があります。

戻り値

戻り値	内容
0x00000000	正常に終了しました。

9.2.3 ETtrans2CreateMdlInfo（MDL 情報の生成）

形式

```
#include  "ETtrans.h"
```

```
int ETtrans2CreateMdlInfo (
  char **pMdlInfo,
  const char *cszMdlFile,
  char *pTC)
```

引数

引数	種別	内容
pMdlInfo	出力	生成した MDL 情報を取得します。
cszMdlFile	入力	ロードしたい MDL ファイル名を指定します。
pTC	入力	使用するスレッド固有情報を指定します。

説明

指定された MDL ファイルをロードして，MDL 情報を生成します。

引数 pMdlInfo で取得した MDL 情報は，変換時に ETtrans2Exec 関数の引数として渡します。

引数 pTC に NULL が指定された場合，この関数を呼び出しているときのログは，ETtrans2Init 関数で指定されたログファイルに出力されます。また，引数 pTC に ETtrans2CreateThreadContext 関数で生成されたスレッド固有情報が指定された場合，この関数を呼び出しているときのログは，スレッド固有情報に指定されたログファイルに出力されます。

戻り値

戻り値	内容
0x00000000	正常に終了しました。
0x02000000	MDL ファイルが検証されていない，MDL ファイルのバージョンが不整合，又は MDL ファイルが不正です。
0x04000000	MDL 情報を生成できませんでした。
0x04000001	指定された MDL ファイルがない，又はファイルへのアクセス権がありません。
0x04000002	メモリ不足です。
0x04000008	スレッド固有情報が不正です。
0x04000010	変換できない MDL ファイルです（対応ライセンスがありません）。

9.2.4 ETtrans2ReleaseMdlInfo（MDL 情報の解放）

形式

```
#include  "ETtrans.h"

int ETtrans2ReleaseMdlInfo (char *pMdlInfo)
```

引数

引数	種別	内容
pMdlInfo	入力	MDL 情報を指定します。

説明

指定された MDL 情報に割り当てられたメモリを解放します。データ変換処理を終了する前に，ETtrans2CreateMdlInfo 関数で生成したすべての MDL 情報に対して，ETtrans2ReleaseMdlInfo 関数を発行し，正常に終了したことを確認する必要があります。

引数 pMdlInfo に既に解放された MDL 情報を渡していた場合は，そのまま正常に終了します。

戻り値

戻り値	内容
0x00000000	正常に終了しました。
0x04000000	MDL 情報を解放できませんでした。
0x04000008	MDL 情報が不正です。

9.2.5 ETtrans2UpdateMdlInfo（MDL 情報の更新）

形式

```
#include  "ETtrans.h"

int ETtrans2UpdateMdlInfo (
  char *pMdlInfo,
  const char *cszMdlFile,
  char *pTC)
```

引数

引数	種別	内容
pMdlInfo	入出力	更新したい MDL 情報のアドレスを指定します。
cszMdlFile	入力	MDL ファイル名を指定します。
pTC	入力	使用するスレッド固有情報を指定します。

説明

指定された MDL 情報にロードされている MDL ファイルをアンロードし，新たに MDL ファイルをロードします。MDL 情報を指定して呼び出している MDL 情報操作関数，又は変換実行関数がある場合は，スレッドの外から同じ MDL 情報を指定して，この関数を呼び出さないでください。

引数 pTC に NULL が指定された場合，この関数を呼び出しているときのログは ETtrans2Init 関数で指定されたログファイルに出力されます。引数 pTC に ETtrans2CreateThreadContext 関数で生成されたスレッド固有情報が指定された場合，この関数を呼び出しているときのログはスレッド固有情報に指定されたログファイルに出力されます。

戻り値

戻り値	内容
0x00000000	正常に終了しました。
0x02000000	MDL ファイルが検証されていない，MDL ファイルのバージョンが不整合，又は MDL ファイルが不正です。
0x04000000	MDL 情報を生成できませんでした。
0x04000001	指定された MDL ファイルがない，又はファイルへのアクセス権がありません。
0x04000002	メモリ不足です。
0x04000008	MDL 情報又はスレッド固有情報が不正です。
0x04000010	変換できない MDL ファイルです（対応ライセンスがありません）。

9.2.6 ETtrans2CreateThreadContext（スレッド固有情報の生成）

形式

```
#include  "ETtrans.h"

int ETtrans2CreateThreadContext (
  char **pTC,
  int iLogFlag,
  const char *cszLogFile,
  int iFileAppendFlag,
  int iFNFlag,
  const char *cszOptList)
```

引数

引数	種別	内容
pTC	出力	スレッド固有情報を取得します。
iLogFlag	入力	ログの出力方法を指定します。 • ET_LG_ON：ログ出力あり • ET_LG_OFF：ログ出力なし
cszLogFile	入力	スレッド固有情報が使用するログファイル名を指定します。
iFileAppendFlag	入力	ファイルの出力方法を指定します。 • ET_FA_UPDATE：新規又は上書き • ET_FA_APPEND：追加書き

引数	種別	内容
iFNFlag	入力	変換するフォーマットを指定するかどうかを指定します。 • ET_OP_FNOFF：指定なし • ET_OP_FNON ：指定あり
cszOptList	入力	コマンドラインで指定できる実行時オプションを指定します。オプションについては，「5.4.1(3)　引数」を参照してください。

説明

スレッド固有情報を生成します。スレッド固有情報にはスレッドのログの出力先や変換実行時のオプションが含まれます。変換実行時のオプションは，ETtrans2CreateThreadContext 関数で生成したスレッド固有情報を使用して呼び出す変換実行時（ETtrans2Exec 関数の呼び出し時）に適用されます。

変換するスレッドごとに ETtrans2CreateThreadContext 関数を発行し，引数 pTC で取得したスレッド固有情報のアドレスを変換実行関数又は ETtrans2CreateThreadContext 関数以外のスレッド固有情報操作関数に渡します。ETtrans2CreateThreadContext 関数で取得したスレッド固有情報のアドレス及び値は変更しないでください。

引数 iLogFlag に ET_LG_OFF を指定した場合，生成されたスレッド固有情報を使用している関数でのログは出力されません。ただし，システム情報ファイル「ettrans.ini」の [Option] セクションでログを出力しないように指定されている場合，引数 iLogFlag の指定に関係なく，ログは出力されません。

引数 cszLogFile にはスレッドで呼び出す関数のログの出力先ファイルを指定します。引数 cszLogFile に NULL を指定した場合，又は引数 cszLogFile に指定したファイルを作成若しくは更新できなかった場合は，ETtrans2Init 関数で指定したログファイルに出力します。

引数 iFileAppendFlag は，変換結果をファイルに出力する場合だけ使用できます（XML 文書の出力は除きます）。ヘッダファイル「ETtrans.h」内で定義されている次の値を指定します。

- ET_FA_UPDATE：新規又は上書き
- ET_FA_APPEND：追加書き

引数 iFNFlag に ET_OP_FNOFF を指定した場合，対象となる MDL ファイル内のすべてのフォーマットを変換します。ET_OP_FNON を指定した場合は，ETtrans2Exec 関数の引数 pAdrsList で指定されたフォーマットだけを変換します（コマンド実行時に -FN オプションを指定した場合と同じです）。

引数 pszOptList には，コマンドラインで指定できる実行時オプションを指定します。-F，-FN，-E，-file，及び -prop は指定できません。

戻り値

戻り値	内容
0x00000000	正常に終了しました。

戻り値	内容
0x01000010※	指定されたログファイルをオープンできないため，ログを初期値のログファイル名で出力，又は標準出力に出力しました。
0x02000000	実行時オプションの指定にエラーがあります。
0x04000000	スレッド固有情報を生成できませんでした。
0x04000002	メモリ不足です。

注※

0x01000010 は，ほかの値と同時に設定される場合があります。

9.2.7 ETtrans2ReleaseThreadContext（スレッド固有情報の解放）

形式

```
#include  "ETtrans.h"

int ETtrans2ReleaseThreadContext (char *pTC)
```

引数

引数	種別	内容
pTC	入力	スレッド固有情報を指定します。

説明

指定されたスレッド固有情報に割り当てられたメモリを解放します。トランスレータが終了する前に，ETtrans2CreateThreadContext 関数で生成したすべてのスレッド固有情報に対して ETtrans2ReleaseThreadContext 関数を発行し，正常終了したことを確認する必要があります。呼び出し中のスレッド固有情報操作関数や変換実行関数がある場合，そこで使用しているものと同じスレッド固有情報を指定しないでください。同じスレッド固有情報を指定して，ETtrans2ReleaseThreadContext 関数を呼び出した場合は，不正な結果となります。

引数 pTC に既に解放されたスレッド固有情報を渡した場合は，そのまま正常に終了します。

戻り値

戻り値	内容
0x00000000	正常に終了しました。
0x04000000	スレッド固有情報を解放できませんでした。
0x04000008	スレッド固有情報が不正です。

9.2.8 ETtrans2UpdateThreadContext（スレッド固有情報の更新）

形式

```
#include  "ETtrans.h"

int ETtrans2UpdateThreadContext(
  char *pTC,
  int iLogFlag,
  const char *cszLogFile,
  int iFileAppendFlag,
  int iFNFlag,
  const char *cszOptList)
```

引数

引数	種別	内容
pTC	入出力	変更したいスレッド固有情報を指定します。
iLogFlag	入力	ログの出力方法を指定します。 • ET_LG_ON：ログ出力あり • ET_LG_OFF：ログ出力なし
cszLogFile	入力	スレッド固有情報が使用するログファイル名を指定します。
iFileAppendFlag	入力	ファイルの出力方法を指定します。 • ET_FA_UPDATE：新規又は上書き • ET_FA_APPEND：追加書き
iFNFlag	入力	変換するフォーマットを指定するかどうかを指定します。 • ET_OP_FNOFF：指定なし • ET_OP_FNON ：指定あり
cszOptList	入力	コマンドラインで指定できる実行時オプションを指定します。オプションについては，「5.4.1(3) 引数」を参照してください。

説明

ログファイル，実行時オプションなどのスレッド固有情報の情報を変更します。この変更は，変更したスレッド固有情報を使用する関数を呼び出したときに使用されます。

エラーが発生した場合，スレッド固有情報は変更されません。

ETtrans2UpdateThreadContext 関数を呼び出したときのログは，引数 pTC で指定したスレッド固有情報に指定されたログファイルに出力されます。呼び出し中のスレッド固有情報操作関数や変換実行関数がある場合，そこで使用しているものと同じスレッド固有情報を指定しないでください。同じスレッド固有情報を指定して，ETtrans2UpdateThreadContext 関数を呼び出した場合は，不正な結果となります。

引数 cszLogFile に NULL を指定した場合，又は引数 cszLogFile に指定したファイルを作成又は更新できなかった場合は，スレッド固有情報に指定されたログファイ

ルは変更されません。
引数 iFileAppendFlag，iFNFlag，及び cszOptList については，「9.2.6 ETtrans2CreateThreadContext（スレッド固有情報の生成）」を参照してください。

戻り値

戻り値	内容
0x00000000	正常に終了しました。
0x01000010※	指定されたログファイルをオープンできないため，ログを初期値のログファイル名で出力，又は標準出力に出力しました。
0x02000000	実行時オプションの指定にエラーがあります。
0x04000000	スレッド固有情報を生成できませんでした。
0x04000002	メモリ不足です。
0x04000008	スレッド固有情報が不正です。

注※
0x01000010 は，ほかの値と同時に設定される場合があります。

9.2.9 ETtrans2Exec（変換実行）

形式

```
#include  "ETtrans.h"

int ETtrans2Exec (
  char *pTC,
  char *pMdlInfo,
  ETTRANSADRSLIST *pAdrsList,
  ETTRANSERRLIST *pErrList,
  void *pPtrToUsrFunc)
```

引数

引数	種別	内容
pTC	入力	スレッド固有情報を指定します。
pMdlInfo	入力	変換に使用する MDL 情報を指定します。
pAdrsList	入出力	変換するフォーマットの入出力データ差し替え情報のリストを指定します。
pErrList	出力	エラー情報リストを取得します。
pPtrToUsrFunc	入力	出口関数に渡すポインタを指定します。

説明

引数 pMdlInfo で指定された MDL 情報に従ってデータ変換をします。
ETtrans2Exec 関数を呼び出しているときのログは，引数 pTC で渡されるスレッド

固有情報を生成したときに指定されたログファイルに出力されます。
引数 pTC には，ETtrans2CreateThreadContext 関数で取得したスレッド固有情報を指定します。複数のスレッドから同じスレッド固有情報を指定して，ETtrans2Exec 関数を呼び出さないでください。
引数 pMdlInfo には，ETtrans2CreateMdlInfo 関数で生成された MDL 情報を指定します。
引数 pAdrsList は，MDL ファイル内に定義した入出力ファイルを，別ファイル又はメモリデータに差し替える場合に指定します。この引数でデータを差し替えるフォーマット名と，対応するデータのファイル名，又はメモリアドレスを指定します。データを差し替えるフォーマット名は複数指定できます。また，差し替えるフォーマットの終了を示すために，ETTRANSADRSLIST 配列の最後の要素の FormatName メンバは NULL を指定する必要があります。
データ差し替えを指定されたフォーマットに対しては，MDL ファイル内で定義された入出力データファイル名は無視されます。入力データと出力データのファイルは，同じファイル名を指定するとエラーとなります。また，MDL ファイル内で定義された入出力データファイル名と同じファイル名を指定した場合もエラーとなります。
構造体 ETTRANSADRSLIST は，ヘッダファイル「ETtrans.h」内で定義されています。

```
typedef struct {
  char  *FormatName; /* 差し替え対象となるフォーマット名 */
  int  DataKind;  /* データ種別 (ET_DK_*) */
  char  *DataAddress; /* 入出力データファイル名（ファイル指定） */
                      /* 入出力データの開始アドレス（メモリ指定） */
  char  *EndAddress; /* 入出力データの終了アドレス（メモリ指定） */
  int  MemoryInitSize; /* 出力データエリア初期サイズ
                                                    （メモリ指定） */
  int  MemoryIncrementSize; /* 出力データエリア増分サイズ
                                                    （メモリ指定） */
} ETTRANSADRSLIST;
```

データ種別 DataKind には，ヘッダファイル「ETtrans.h」内で定義されている次の値のどれかを指定します。

- ET_DK_FILE：データ種別がファイルの場合
- ET_DK_MEMORY：データ種別がメモリの場合

ET_DK_MEMORY を指定した場合は，入出力データの開始アドレス DataAddress と終了アドレス EndAddress を指定してください。ただし，出力フォーマットの場合は，開始アドレスとして NULL アドレスを指定できます。この場合，トランスレータが必要なデータエリアを確保し，その先頭アドレスを入出力データの開始アドレス DataAddress へ設定して返します。トランスレータが確保したエリアは，次の ETtrans2Exec 関数の実行，又は ETtrans2ReleaseThreadContext 関数の実行前まで有効です。確保するデータエリアサイズは，pAdrsList の MemoryInitSize，又は MemoryIncrementSize で指定します。pAdrsList の該当する値に 0 が指定され

た場合は，システム情報ファイル「ettrans.ini」内の [MemoryData] セクションで定義された値を用います。pAdrsList，システム情報ファイルの両方に定義がない場合は，初期割り当てサイズと割り当て増分サイズの両方に 1,024（バイト）を仮定します。
出力フォーマットに対しては，入出力データの終了アドレス EndAddress にデータの最終アドレスを返します。ただし，出力データが指定されたアドレスを超えてエラーのために出力されない場合は，ヘッダファイル「ETtrans.h」で定義されている次の値を設定して返します。

- ET_AD_OVER ：メモリアドレスオーバ
- ET_AD_NOTOUT ：未出力

引数 pErrList へは変換実行時のエラー情報を出力します。エラー情報がない場合には，エラー情報リスト pErrList のデータ数に 0 を返します。取り出した内容は次の ETtrans2Exec 関数の実行，又は ETtrans2ReleaseThreadContext 関数の実行前まで有効です。構造体 ETTRANSERRLIST は，ヘッダファイル「ETtrans.h」内で定義されています。

```
typedef struct {
  int  Size;   /* エラー情報データ数 */
  ETTRANSERRDATA  *ErrData; /* エラー情報データ(配列) */
} ETTRANSERRLIST;

typedef struct {
  int  MessageNo; /* メッセージ番号 */
  int  ErrorLevel; /* エラーレベル */
  int  NumericData; /* 数値情報 */
  char  *Information; /* 文字列情報 */
} ETTRANSERRDATA;
```

各エラー情報のデータには，変換処理で発生したワーニングレベル以上のエラー情報が発生順に設定されます。エラーレベルは，トランスレータの戻り値と同じ意味を持ちます。数値情報と文字列情報は，ログファイルへ出力する付加情報で，次の内容を出力します。付加情報がない場合は，数値情報は 0，文字列情報は NULL が設定されます。

表 9-2　ログファイルに出力される数値情報，文字情報の内容

メッセージ番号	数値情報	文字列情報	備考
9	出力済みのグループ数	なし	グループ単位の出力指定がある場合のエラー
55，56	出口関数の戻り値，例外中のエラーコード（Java 言語の出口関数の場合）	ユーザ組み込み関数名	出口関数のエラー
70，71，72	0	ユーザ組み込み関数名	Java 言語の出口関数のエラー

メッセージ番号	数値情報	文字列情報	備考
7，8，23 ～ 36，50，57，67，78	コンポーネントのアドレス	コンポーネント名	コンポーネントに対するエラー
21	0	オプション	オプション不正エラー
	1	フォーマット名	
	2	なし（メモリアドレス不正の場合）	

注

メッセージ番号の内容については，「付録 B.3　メッセージ」を参照してください。

引数 pPtrToUsrFunc には，出口関数に渡す任意のアドレスを指定します。トランスレータは，システム情報ファイルの [Userfunc_Option] セクションで，USE02IF オプションを指定された出口関数を呼び出すときに，ETtrans2Exec 関数で指定された任意のアドレスを出口関数のポインタ型として入力引数へ追加して渡します。ポインタ型の引数については，「11.3.2 ［Userfunc_Mapfunc］セクション」の「表 11-2　ユーザ組み込み関数で使用する型」を参照してください。

トランスレータは指定されたアドレスの更新及びチェックはしません。USE02IF オプションを指定されていない出口関数は，引数 pPtrToUsrFunc の指定は無視されます。

戻り値

トランスレータでのデータ変換の戻り値を同じです。ettrans コマンドの戻り値については，「5.4.2　ettrans コマンドの戻り値」を参照してください。

9.2.10　ETtrans2GetDlProperty（FDL ファイル又は MDL ファイルのプロパティ情報の取得）

形式

```
#include  "ETtrans.h"

int ETtrans2GetDlProperty (
  const char *cszFileName,
  ETTRANSDLPROPINFO *pDlPropInfo)
```

引数

引数	種別	内容
cszFileName	入力	FDL ファイル又は MDL ファイル名を指定します。
pDlPropInfo	出力	DL プロパティ情報を取得します。

説明

指定された FDL ファイル又は MDL ファイルから DL プロパティ情報を取得します。取得した DL プロパティ情報の値は，ETtrans2ReleaseDlProperty 関数を呼び出すまで有効です。構造体 ETTRANSDLPROPINFO は，ヘッダファイル「ETtrans.h」内で次のように定義されています。

```
typedef struct _ETTRANSDLPROPINFO {
  char *szUserVersion;    /* ユーザバージョン */
  char *szUserComment;    /* ユーザコメント */
  long lSysCreatedDate;    /* 作成日時 */
  char *szSysCreatedPpVersion;  /* 作成製品バージョン */
  long lSysLastUpdatedDate;  /* 更新日時 */
  char *szSysLastUpdatedPpVersion; /* 更新製品バージョン */
  unsigned long lSysDlVersion;  /* DLファイルバージョン */
} ETTRANSDLPROPINFO;
```

ETtrans2GetDlProperty 関数を呼び出しているときは，ログは出力されません。
ユーザバージョン szUserVersion とユーザコメント szUserComment は，実行環境の文字コードに変換して返します。ただし，ワークステーションの OS では，LANG 環境変数が ” C ” に設定されていた場合は，FDL ファイル又は MDL ファイルを作成したときに格納された文字列をそのまま返します。この二つの情報が設定されていない場合は，対応する構造体メンバに NULL が設定されます。
作成日時 lSysCreatedDate，更新日時 lSysLastUpdatedDate の値には万国標準時（UCT）の 1970 年 1 月 1 日の 00:00:00 から作成日時，更新日時までの通算経過秒が設定されます。
作成製品バージョン szSysCreatedPpVersion，更新製品バージョン szSysLastUpdatedPpVersion には，それぞれ作成又は更新に使用した Interschema のバージョンが次の形式で返されます。

VVRRSS

VV：バージョン番号
RR：リビジョン番号
SS：限定・修正コード

DL ファイルバージョン lSysDlVersion には FDL バージョン又は MDL バージョンが設定されます。
古いバージョンの FDL 又は MDL（電子データ変換ツール，又は Interschema バージョン 1 で作成したもの）に対して ETtrans2GetDlProperty 関数を実行した場合は，有効な情報は lSysLastUpdatedDate，lSysDlVersion だけです。

戻り値

戻り値	内容
0x00000000	正常に終了しました。
0x02000000	不正な FDL 又は MDL が指定されました。

戻り値	内容
0x02000001	コード変換で不正文字コードを検出しました。
0x04000000	システムエラーがあります（パラメタエラーを含む）。
0x04000001	ファイルの入出力にエラーがあります。
0x04000002	メモリ不足です。

9.2.11 ETtrans2ReleaseDlProperty（FDL ファイル又は MDL ファイルのプロパティ情報の解放）

形式

```
#include  "ETtrans.h"

int ETtrans2ReleaseDlProperty (ETTRANSDLPROPINFO *pDlPropInfo)
```

引数

引数	種別	内容
pDlPropInfo	入力	DL プロパティ情報を指定します。

説明

指定された DL プロパティ情報の構造体の各 char* 型メンバ領域を解放します。ETtrans2GetDlProperty 関数で取得したすべての DL プロパティ情報に対して，この関数を呼び出して解放処理をする必要があります。この関数を呼び出しているときは，ログは出力されません。

戻り値

戻り値	内容
0x00000000	正常に終了しました。
0x04000000	引数にエラーがあります。

9.3 データ変換処理 API（C 言語）の使用例

この節では，データ変換処理 API（C 言語）の使用例を示します。

データ変換処理 API（C 言語）の使用例には，次の三つの例があります。

- シングルスレッドの場合
- マルチスレッド（スレッド内で MDL 情報を生成しない）の場合
- マルチスレッド（スレッド内で MDL 情報を生成する）の場合

（1）シングルスレッドの場合

```
#include <stdio.h>
#include "ETtrans.h"

static void print_message(int iCode, char *msg,
                          ETTRANSERRLIST *ErrList);

int main()
{
  int iRet = 0;
  ETTRANSADRSLIST AdrsList[3];
  ETTRANSERRLIST ErrList;
  char *pTC = NULL;
  char *pMI = NULL;
                          /* アドレスリストの設定 */
  AdrsList[0].FormatName = "IN";
  AdrsList[0].DataKind = ET_DK_FILE;
  AdrsList[0].DataAddress = "input.txt";
  AdrsList[1].FormatName = "OUT";
  AdrsList[1].DataKind = ET_DK_FILE;
  AdrsList[1].DataAddress = "output.txt";
  AdrsList[2].FormatName = NULL;
                          /* トランスレータ初期化 */
  iRet = ETtrans2Init(NULL);
  if (iRet) {
    print_message(iRet, "ETtrans2Init", NULL);
    goto EXIT;
  }
                          /* スレッド固有情報生成 */
  iRet = ETtrans2CreateThreadContext(
    &pTC, ET_LG_ON,NULL, ET_FA_UPDATE, ET_OP_FNOFF, "-IERR");
  if (iRet) {
    print_message(iRet, "ETtrans2CreateThreadContext", NULL);
    goto END;
  }
                          /* MDLローディング */
  iRet = ETtrans2CreateMdlInfo(&pMI, "SAMPLE.mdl", NULL);
  if (iRet) {
    print_message(iRet, "ETtrans2CreateMdlInfo", NULL);
    goto END;
  }
                          /* 変換実行 */
```

```
  iRet = ETtrans2Exec(pTC, pMI, AdrsList, &ErrList, NULL);
  if (iRet) {
    print_message(iRet, "ETtrans2Exec", &ErrList);
    goto END;
  }

END:
  if (pMI) {              /* MDLアンロード */
    iRet = ETtrans2ReleaseMdlInfo(pMI);
    if (iRet) {
      print_message(iRet, "ETtrans2ReleaseMdlInfo", NULL);
    }
  }

  if (pTC) {              /* スレッド固有情報解放 */
    iRet = ETtrans2ReleaseThreadContext(pTC);
    if (iRet) {
    print_message(iRet, "ETtrans2ReleaseThreadContext", NULL);
  }
  }
                          /* 終了処理 */
  iRet = ETtrans2End();
  if (iRet) {
    print_message(iRet, "ETtrans2End", NULL);
  }

EXIT:
  printf("--> Return Code = 0x%08x¥n", iRet);
  return iRet;
}

/*
 * エラー情報出力関数
 */
static void print_message(int iCode, char *msg, ETTRANSERRLIST
*ErrList)
{
  int i;
  ETTRANSERRDATA *edata;

  printf("### ERROR %s %d ###¥n", msg, iCode);
  if (ErrList) {
    printf(" Error Size = %d¥n", ErrList->Size);
    for (i = 0; i < ErrList->Size; i++) {
      edata = &ErrList->ErrData[i];
      printf("¥t%02d (0x%08x) num=%d",
        edata->MessageNo, edata->ErrorLevel, edata->NumericData);
      if (edata->Information) {
        printf(" info=%s¥n", edata->Information);
      } else {
        printf("¥n");
      }
    }
  }
}
```

（2）マルチスレッド（スレッド内で MDL 情報を生成しない）の場合

```
#include <stdio.h>
#include <string.h>
#ifdef WIN32
  #include <process.h>
  #include <windows.h>
#else
  #include <pthread.h>
#endif
#include "ETtrans.h"

#define THRD_NUM  3  /* スレッドの数 */

/* スレッドに渡す情報を格納する構造体 */
typedef struct _THD_INF {
  char *pMI; /* MDL情報 */
  char *pTC; /* スレッド固有情報 */
  char InputDataFile[256]; /* 入力データファイル名 */
  char OutputDataFile[256]; /* 出力データファイル名 */
  int iTid; /* スレッド番号 */
} THD_INF;

#ifdef WIN32
  DWORD  WINAPI ettrans_thrd(LPVOID);
#else
  void *ettrans_thrd(void*);
#endif

/*
 * main : メイン関数
 */
int main()
{
  int iRet = 0;
  int i;
  char *pMI = NULL;
  char *pTC_Ary[THRD_NUM];
  THD_INF thd_inf_Ary[THRD_NUM];
  char numwk[12];
#ifdef WIN32
  DWORD thrdid[THRD_NUM];
  HANDLE hthrd[THRD_NUM];
#else
  int ret_thread;
  pthread_t hthrd[THRD_NUM];
#endif
  for (i = 0 ; i < THRD_NUM; i++) {
    pTC_Ary[i] = NULL;
  }
                /* トランスレータ初期化 */
  iRet = ETtrans2Init(NULL);
  if (iRet) goto EXIT;
                /* MDLローディング */
  iRet = ETtrans2CreateMdlInfo(&pMI, "SAMPLE.mdl", NULL);
```

```
  if (iRet) goto END;
                  /* スレッド固有情報生成 */
  for (i = 0 ; i < THRD_NUM; i++) {
    iRet = ETtrans2CreateThreadContext(&pTC_Ary[i], ET_LG_ON, NULL,
ET_FA_UPDATE, ET_OP_FNOFF, "-IERR");
    if (iRet) goto END;
  }
                  /* スレッド生成 & 変換実行 */
  for (i = 0 ; i < THRD_NUM; i++) {
    sprintf(numwk, "%d", i + 1);
    strcpy(thd_inf_Ary[i].InputDataFile, "input");
    strcat(thd_inf_Ary[i].InputDataFile, numwk);
    strcat(thd_inf_Ary[i].InputDataFile, ".txt");
    strcpy(thd_inf_Ary[i].OutputDataFile, "output");
    strcat(thd_inf_Ary[i].OutputDataFile, numwk);
    strcat(thd_inf_Ary[i].OutputDataFile, ".txt");
    thd_inf_Ary[i].iTid = i + 1;
    thd_inf_Ary[i].pMI = pMI;
    thd_inf_Ary[i].pTC = pTC_Ary[i];

#ifdef WIN32
  hthrd[i] = CreateThread(NULL, 0, ettrans_thrd,
(LPVOID)&thd_inf_Ary[i],
                          0, &thrdid[i]);
#else
  ret_thread = pthread_create(&hthrd[i], NULL, ettrans_thrd,
                              &thd_inf_Ary[i]);
#endif
  }

                  /* スレッド終了待合せ */
#ifdef WIN32
  for (i = 0; i < THRD_NUM; i++) {
    WaitForSingleObject(hthrd[i], INFINITE);
  }
  for ( i = 0 ; i < THRD_NUM; i++ ){
    CloseHandle(hthrd[i]);
  }
#else
  for ( i = 0 ; i < THRD_NUM; i++ ){
    pthread_join(hthrd[i], NULL);
  }
#endif

END:
  if (pMI) {            /* MDLアンロード */
    ETtrans2ReleaseMdlInfo(pMI);
  }

  for (i = 0 ; i < THRD_NUM; i++) {
    if (pTC_Ary[i]) { /* スレッド固有情報解放 */
      ETtrans2ReleaseThreadContext(pTC_Ary[i]);
    }
  }

                      /* 終了処理 */
 ETtrans2End();
```

```
EXIT:
  printf("*** END - 0x%08x ***¥n", iRet);
  return(iRet);
}


/*
 * ettrans_thrd : スレッド対応関数
 */
#ifdef WIN32
DWORD WINAPI ettrans_thrd(LPVOID Inf_Arg)
#else
void *ettrans_thrd(void *Inf_Arg)
#endif
{
  int iRet = 0;
  int iTid = 0;
  THD_INF thd_inf;
  ETTRANSADRSLIST AdrsList[3];
  ETTRANSERRLIST ErrList;

  thd_inf = *(THD_INF*)Inf_Arg;
  iTid = thd_inf.iTid;

           /* 引数の設定 */
  AdrsList[0].FormatName = "IN";
  AdrsList[0].DataKind = ET_DK_FILE;
  AdrsList[0].DataAddress = thd_inf.InputDataFile;
  AdrsList[1].FormatName = "OUT";
  AdrsList[1].DataKind = ET_DK_FILE;
  AdrsList[1].DataAddress = thd_inf.OutputDataFile;
  AdrsList[2].FormatName = NULL;

          /* 変換実行 */
  iRet = ETtrans2Exec(thd_inf.pTC, thd_inf.pMI, AdrsList, &ErrList,
NULL);
  printf("Thread %08d End - Return Code 0x%08x¥n", iTid, iRet);
#ifdef WIN32
  ExitThread(0);
#else
  pthread_exit((void *)0);
#endif
  return 0;
}
```

（3）マルチスレッド（スレッド内で MDL 情報を生成する）の場合

```
#include <stdio.h>
#include <string.h>
#ifdef WIN32
  #include <process.h>
  #include <windows.h>
#else
```

```
  #include <pthread.h>
#endif
#include "ETtrans.h"

#define THRD_NUM  3  /* スレッドの数 */

/* スレッドに渡す情報を格納する構造体 */
typedef struct _THD_INF {
  char *pTC; /* スレッド固有情報 */
  char InputDataFile[256]; /* 入力データファイル名 */
  char OutputDataFile[256]; /* 出力データファイル名 */
  int iTid; /* スレッド番号 */
} THD_INF;

#ifdef WIN32
  DWORD  WINAPI ettrans_thrd(LPVOID);
#else
  void *ettrans_thrd(void*);
#endif

/*
 * main : メイン関数
 */
int main()
{
  int iRet = 0;
  int i;
  char *pTC_Ary[THRD_NUM];
  THD_INF thd_inf_Ary[THRD_NUM];
  char numwk[12];
#ifdef WIN32
  DWORD thrdid[THRD_NUM];
  HANDLE hthrd[THRD_NUM];
#else
  int ret_thread;
  pthread_t hthrd[THRD_NUM];
#endif
  for (i = 0 ; i < THRD_NUM; i++) {
    pTC_Ary[i] = NULL;
  }
                    /* トランスレータ初期化 */
  iRet = ETtrans2Init(NULL);
  if (iRet) goto EXIT;
                    /* スレッド固有情報生成 */
  for (i = 0 ; i < THRD_NUM; i++) {
    iRet = ETtrans2CreateThreadContext(&pTC_Ary[i], ET_LG_ON, NULL,
      ET_FA_UPDATE, ET_OP_FNOFF, "-IERR");
  if (iRet) goto END;
  }
                   /* スレッド生成 & 変換実行 */
  for (i = 0 ; i < THRD_NUM; i++) {
    sprintf(numwk, "%d", i + 1);
    strcpy(thd_inf_Ary[i].InputDataFile, "input");
    strcat(thd_inf_Ary[i].InputDataFile, numwk);
    strcat(thd_inf_Ary[i].InputDataFile, ".txt");
    strcpy(thd_inf_Ary[i].OutputDataFile, "output");
    strcat(thd_inf_Ary[i].OutputDataFile, numwk);
    strcat(thd_inf_Ary[i].OutputDataFile, ".txt");
```

```
    thd_inf_Ary[i].iTid = i + 1;
    thd_inf_Ary[i].pTC = pTC_Ary[i];

#ifdef WIN32
  hthrd[i] = CreateThread(NULL, 0, ettrans_thrd,
             (LPVOID)&thd_inf_Ary[i], 0, &thrdid[i]);
#else
  ret_thread = pthread_create(&hthrd[i], NULL, ettrans_thrd,
&thd_inf_Ary[i]);
#endif
  }

                /* スレッド終了待合せ */
#ifdef WIN32
  for (i = 0; i < THRD_NUM; i++) {
    WaitForSingleObject(hthrd[i], INFINITE);
  }
  for ( i = 0 ; i < THRD_NUM; i++ ){
    CloseHandle(hthrd[i]);
  }
#else
  for ( i = 0 ; i < THRD_NUM; i++ ){
    pthread_join(hthrd[i], NULL);
  }
#endif

END:
  for (i = 0 ; i < THRD_NUM; i++) {
    if (pTC_Ary[i]) {          /* スレッド固有情報解放 */
      ETtrans2ReleaseThreadContext(pTC_Ary[i]);
    }
  }

               /* 終了処理 */
  ETtrans2End();

EXIT:
  printf("*** END - 0x%08x ***¥n", iRet);
  return(iRet);
}


/*
 * ettrans_thrd : スレッド対応関数
 */
#ifdef WIN32
DWORD WINAPI ettrans_thrd(LPVOID Inf_Arg)
#else
void *ettrans_thrd(void *Inf_Arg)
#endif
{
  int iRet = 0;
  int iTid = 0;
  char *pMI = NULL;
  THD_INF thd_inf;
  ETTRANSADRSLIST AdrsList[3];
  ETTRANSERRLIST ErrList;
```

```
  thd_inf = *(THD_INF*)Inf_Arg;
  iTid = thd_inf.iTid;

       /* 引数の設定 */
  AdrsList[0].FormatName = "IN";
  AdrsList[0].DataKind = ET_DK_FILE;
  AdrsList[0].DataAddress = thd_inf.InputDataFile;
  AdrsList[1].FormatName = "OUT";
  AdrsList[1].DataKind = ET_DK_FILE;
  AdrsList[1].DataAddress = thd_inf.OutputDataFile;
  AdrsList[2].FormatName = NULL;

       /* MDLローディング */
  iRet = ETtrans2CreateMdlInfo(&pMI, "SAMPLE.mdl", NULL);
  if (iRet) goto END;
       /* 変換実行 */
  iRet = ETtrans2Exec(thd_inf.pTC, pMI, AdrsList, &ErrList, NULL);
END:
  if (pMI) {   /* MDLアンロード */
    ETtrans2ReleaseMdlInfo(pMI);
  }

  printf("Thread %08d End - Return Code 0x%08x¥n", iTid, iRet);
#ifdef WIN32
  ExitThread(0);
#else
  pthread_exit((void *)0);
#endif
  return 0;
}
```

9.4 データ変換処理 API（C 言語）を使用したプログラムの作成

この節では，データ変換処理 API（C 言語）を使用してプログラムを作成するときの手順について説明します。

9.4.1 データ変換処理 API（C 言語）を使用したプログラムの作成（Windows の場合）

データ変換処理 API（C 言語）を使用したプログラムを作成して，プログラムを実行するために必要な手順について説明します。

（1） プログラムの作成

データ変換処理 API（C 言語）を使用してプログラムを作成する場合，トランスレータのインポートライブラリとリンクを取る必要があります。インポートライブラリとリンクを取るには，作成するプログラムのリンクオプションで，トランスレータで提供するインポートライブラリ（ETapir.lib）を取り込みます。ETapir.lib の格納ディレクトリについては，Interschema のリリースノートを参照してください。

（2） プログラムの実行

作成したプログラムを実行する場合は，トランスレータで提供する DLL（ETapir.dll）が必要です。作成したプログラムを ETapir.dll と同じディレクトリにコピーするか，又は ETapir.dll の格納ディレクトリを環境変数 PATH に設定してください。ETapir.dll の格納ディレクトリについては，Interschema のリリースノートを参照してください。

9.4.2 データ変換処理 API（C 言語）を使用したプログラムの作成（ワークステーションの OS の場合）

データ変換処理 API（C 言語）を使用してプログラムを作成する場合，共用ライブラリをリンクしてください。共用ライブラリは，次のとおりです。

- HP-UX(PA-RISC) の場合：libetapi.sl
- HP-UX(IPF) の場合（32bit 版ライブラリ）：libetapi.so
- HP-UX(IPF) の場合（64bit 版ライブラリ）：libetapi64.so
- AIX の場合：libetapi.a
- Solaris の場合：libetapi.so

共用ライブラリの格納場所については，Interschema のリリースノートを参照してください。

その後，作成しておいた makefile を make コマンドで実行して，ソースファイルをオブジェクトファイル（プログラム）に変換します。

makefile の作成例は，サンプルファイルを参考にしてください。サンプルファイルの格納先については，Interschema のリリースノートを参照してください。

なお，各関数の引数に指定できるファイル名の長さは，256 バイトまでです。プログラム作成時は注意してください。

10 データ変換処理 API（Java 言語）

この章では，データ変換処理 API（Java 言語）について説明します。

10.1 データ変換処理 API（Java 言語）の概要

Interschema は，Java 言語の実行環境で，ユーザ業務プログラムからトランスレータを起動してデータ変換を行うために必要な，データ変換処理 API（Java 言語）を提供します。

ただし，データ変換処理 API（Java 言語）を利用できるユーザ業務プログラムは，Cosminexus Component Conteiner 上でフォーマットを変換する目的で作成された Servlet に限ります。Servlet 以外の Java 言語で作成された一般のプログラムからは，データ変換処理 API（Java 言語）を利用することはできません。

また，データ変換処理 API（Java 言語）は，内部で Java Native Interface（以降，JNI と呼びます）を使用しているため，データ変換処理 API（Java 言語）を使用して作成したユーザ業務プログラムを，Enterprise Java Beans（EJB）から直接呼び出すことはできません。

この節では，データ変換処理 API（Java 言語）の機能概要，パッケージ構成，各パッケージに含まれる Java クラスの仕様，及び使用時の注意事項について説明します。データ変換処理 API（Java 言語）の関数を使用する前に，必ずお読みください。

10.1.1 データ変換処理 API（Java 言語）の機能概要

データ変換処理 API（Java 言語）は，JNI 経由でデータ変換処理 API（C 言語）を呼び出すことによって，Java 言語実行環境にデータ変換処理 API（C 言語）と同等の機能を提供します。さらに，Interschema が提供する抽象クラス TranslateData を継承したクラスを作成すると，任意のリソースから変換データを入出力することができます。

10.1.2 データ変換処理 API（Java 言語）のパッケージ構成

データ変換処理 API（Java 言語）は，次の二つのパッケージから構成されます。

- jp.co.Hitachi.soft.interschema2
- jp.co.Hitachi.soft.interschema2.exitfunc

jp.co.Hitachi.soft.interschema2 パッケージ，及び
jp.co.Hitachi.soft.interschema2.exitfunc パッケージは，次のディレクトリに格納されています。

Interschema のインストールディレクトリ/classes

jp.co.Hitachi.soft.interschema2 パッケージ，及び
jp.co.Hitachi.soft.interschema2.exitfunc パッケージは，Jar ファイル「ETransJ2.jar」で定義されています。

jp.co.Hitachi.soft.interschema2 パッケージ，及び
jp.co.Hitachi.soft.interschema2.exitfunc パッケージを利用するためには，環境変数「CLASSPATH」に Jar ファイル「ETtransJ2.jar」のパスを設定する必要があります。ワークステーションの OS で Interschema を使用する場合，Jar ファイル「ETtransJ2.jar」の設定については，「5.1.4　クラスパスの設定（ワークステーションの OS の場合）」を参照してください。Windows で Interschema を使用する場合，Jar ファイル「ETtransJ2.jar」の設定については，「5.2.1　クラスパスの設定（Windows の場合）」を参照してください。

10.1.3　jp.co.Hitachi.soft.interschema2 パッケージのクラスの概要

jp.co.Hitachi.soft.interschema2 パッケージに含まれる Java クラスの一覧を次に示します。

表 10-1　jp.co.Hitachi.soft.interschema2 パッケージの Java クラス一覧

項番	クラス名	機能概要
1	Translator	変換処理を行うトランスレータを定義します。
2	MDLInfo	トランスレータの動作を定義する変換情報を定義します。
3	DLProperty	DL プロパティ情報を定義します。
4	Option	トランスレータの変換実行時オプションを定義します。
5	TranslateData	変換対象データを定義します。
6	FileData	ファイル形式の変換対象データを定義します。
7	StringData	文字列データ形式の変換対象データを定義します。
8	InputStreamData	入力ストリーム形式の変換対象データを定義します。
9	OutputStreamData	出力ストリーム形式の変換対象データを定義します。
10	TranslatorException	データ変換処理 API（Java 言語）実行時に発生したエラー例外を定義します。
11	ExtraErrorData	拡張エラー情報を定義します。
12	UserException	ユーザ定義クラスで発生したエラー例外を定義します。

10.1.4　jp.co.Hitachi.soft.interschema2.exitfunc パッケージのクラスの概要

jp.co.Hitachi.soft.interschema2.exitfunc パッケージに含まれる Java クラスの一覧を次に示します。

表 10-2　jp.co.Hitachi.soft.interschema2.exitfunc パッケージの Java クラス一覧

項番	クラス名	機能概要
1	ENVString	Interschema の ET_ENVSTRING 型に対応する Java クラスです。出口関数の引数に使用します。
2	DateTime	Interschema の ET_DTM 型に対応する Java クラスです。出口関数の引数及び戻り値に使用します。
3	ExitFuncWarning	出口関数で発生したワーニングを定義します。
4	ExitFuncException	出口関数で発生したエラー例外を定義します。

10.1.5　データ変換処理 API（Java 言語）使用時の注意事項

データ変換処理 API（Java 言語）の関数を使用する場合は，次の点に注意してください。

- マルチスレッドでデータ変換を行う場合，Translator クラスのインスタンスをスレッド数だけ生成してください。一つのインスタンスをスレッド間で共有してデータを変換することはできません。
- データ変換に使用中の MDLInfo クラスのインスタンスに対して，reload メソッドを呼び出すことはできません。
- TranslateData クラス，及びその派生クラスのインスタンスを，スレッド間で共有してデータを変換することはできません。
- ガーベジコレクションによるメモリ解放の効果を大きくするために，不要になったインスタンスには null を代入することを推奨します。

10.2　jp.co.Hitachi.soft.interschema2 パッケージのクラスの仕様

この節では，jp.co.Hitachi.soft.interschema2 パッケージに含まれる Java クラスの仕様について説明します。この節の記述形式を次に示します。

継承

Java クラスの継承関係を示します。

形式

Java クラス，フィールド，コンストラクタ，及びメソッドの記述形式を示します。

引数

Java クラス，コンストラクタ，及びメソッドの引数を示します。

説明

Java クラス，フィールド，コンストラクタ，及びメソッドの機能について説明します。

戻り値

Java クラス，及びメソッドの戻り値を示します。

例外

Java クラス，コンストラクタ，及びメソッドの例外を示します。また，例外が発生する条件について説明します。

10.2.1　Translator クラス

ここでは，Translator クラスの仕様について説明します。

（1）Translator クラスの概要

継承

```
java.lang.Object
|
+-- jp.co.Hitachi.soft.interschema2.Translator
```

形式

```
public final class Translator
extends java.lang.Object
```

説明

変換処理を行うトランスレータを定義します。トランスレータは，マルチスレッド

に対応しているため，プロセス内で Translator クラスのインスタンスを複数作成して，変換処理を並列に実行できます。ただし，一つの Translator クラスのインスタンスに対して，複数のスレッドから同時にアクセスすることはできません。

コンストラクタの概要

形式	機能
public Translator(String logFileName,Option option)	指定されたログファイル，及び指定された変換オプションで，トランスレータを生成します。
public Translator(String logFileName)	指定されたログファイル，及びデフォルトの変換オプションで，トランスレータを生成します。
public Translator(Option option)	デフォルトのログファイル，及び指定された変換オプションで，トランスレータを生成します。
public Translator()	デフォルトのログファイル，及びデフォルトの変換オプションで，トランスレータを生成します。

メソッドの概要

形式	機能
public int exec(MDLInfo mdlInfo, List inputData,List outputData,java.lang.Object userObj)	指定された MDL 情報に従って，データ変換を行います。変換中に呼び出す Java 言語の出口関数に，最後の引数に指定されたオブジェクトを渡します。
public int exec(MDLInfo mdlInfo, List inputData,List outputData)	指定された MDL 情報に従って，データ変換を行います。
public Option getOption()	現在の変換オプションを取得します。
public void setOption(Option option)	指定された変換オプションを設定します。

（2） コンストラクタの詳細

（a） Translator

形式

```
public Translator(String logFileName,Option option)
    throws TranslatorException
```

引数

引数	内容
logFileName	ログの出力先ファイル名を指定します。
option	オプションを指定します。

説明

指定されたログファイル，及び指定された変換オプションで，トランスレータを生成します。

logFileName に null が指定された場合，又は logFileName に指定されたファイルを作成・更新できなかった場合は，デフォルトのログファイル（*Interschema のインストールディレクトリ*/log/errlog.txt）にログを出力します。デフォルトのログファイルを作成・更新できなかった場合は，標準エラー出力にログを出力します。

option に null が指定された場合は，デフォルトのオプションが指定されたものとします。オプションのデフォルト値については，「10.2.4　Option クラス」を参照してください。

例外

例外	内容
TranslatorException	ライブラリの初期化又はトランスレータの生成に失敗した場合に発生します。

(b) Translator

形式

```
public Translator(String logFileName)
    throws TranslatorException
```

引数

引数	内容
logFileName	ログの出力先ファイル名を指定します。

説明

指定されたログファイル，及びデフォルトの変換オプションで，トランスレータを生成します。変換オプションに null を指定してトランスレータを生成する場合と等価です。

例外

例外	内容
TranslatorException	ライブラリの初期化又はトランスレータの生成に失敗した場合に発生します。

(c) Translator

形式

```
public Translator(Option option)
    throws TranslatorException
```

引数

引数	内容
option	オプションを指定します。

説明

デフォルトのログファイル，及び指定された変換オプションで，トランスレータを生成します。ログファイルに null を指定してトランスレータを生成する場合と等価です。

例外

例外	内容
TranslatorException	ライブラリの初期化又はトランスレータの生成に失敗した場合に発生します。

（d） Translator

形式

```
public Translator()
    throws TranslatorException
```

引数

なし

説明

デフォルトのログファイル，及びデフォルトの変換オプションで，トランスレータを生成します。ログファイルと変換オプションに null を指定してトランスレータを生成する場合と等価です。

例外

例外	内容
TranslatorException	ライブラリの初期化又はトランスレータの生成に失敗した場合に発生します。

（3） メソッドの詳細

（a） exec

形式

```
public int exec(MDLInfo mdlInfo, List inputData, List
outputData, java.lang.Object userObj)
    throws TranslatorException,UserException
```

引数

引数	内容
mdlInfo	変換に使用する MDL 情報を指定します。
inputData	入力する変換対象フォーマット情報のリストを指定します。
outputData	出力する変換対象フォーマット情報のリストを指定します。
userObj	出口関数に渡すオブジェクトを指定します。

説明

指定された MDL 情報に従ってデータ変換を行います。

変換対象フォーマット情報は，MDL 情報内で指定されている入出力ファイルを，別ファイル又はメモリデータに変換する場合に指定します。変換対象フォーマットは複数指定できます。指定された変換対象フォーマットに対しては，MDL 内で設定された入出力データファイル名を無視します。

mdlInfo に null が指定された場合，パラメタエラーとして TranslatorException が発生します。

inputData 又は outputData に null が指定された場合，MDL 情報内で指定されている入出力ファイルを変換します。inputData 又は outputData 内に null の要素が含まれる場合は，その要素を無視します。

userObj に指定されたオブジェクトは，この関数呼び出しの延長で，Java 言語の出口関数が呼ばれた時に，第 1 引数にそのまま渡されます。null を指定することもできます。

戻り値には，JNI 経由で呼び出す ETtrans2Exec 関数のリタンコードが返されますが，変換の途中でデータ変換を終了するレベルのエラー（ETtrans2Exec 関数のリタンコードで 0x02yyyyyy 又は 0x04zzzzzz に該当するエラー）が発生した場合は，TranslatorException 例外がスローされて，戻り値は取得できません。

戻り値

JNI 経由で呼び出す ETtrans2Exec 関数のリタンコードが返されます。

例外

例外	内容
TranslatorException	MDL 情報に null が指定された場合，又は変換に失敗した場合に発生します。
UserException	ユーザが定義した変換対象データでエラーが発生した場合に発生します。

(b) exec

形式

```
public int exec(MDLInfo mdlInfo, List inputData, List
outputData)
```

```
        throws TranslatorException, UserException
```

引数

引数	内容
mdlInfo	変換に使用する MDL 情報を指定します。
inputData	入力する変換対象フォーマット情報のリストを指定します。
outputData	出力する変換対象フォーマット情報のリストを指定します。

説明

指定された MDL 情報に従ってデータ変換を行います。引数が三つの形式の exec を，出口関数に渡すオブジェクト（引数 userObj）に null を指定して呼び出した場合と等価です。

戻り値

JNI 経由で呼び出す ETtrans2Exec 関数のリタンコードが返されます。

例外

例外	内容
TranslatorException	MDL 情報に null が指定された場合，又は変換に失敗した場合に発生します。
UserException	ユーザが定義した変換対象データでエラーが発生した場合に発生します。

(c) getOption

形式

```
public Option getOption()
```

引数

なし

説明

現在のオプションを取得します。

戻り値

オプションが返されます。

(d) setOption

形式

```
public void setOption(Option option)
    throws TranslatorException
```

引数

引数	内容
option	オプションを指定します。

説明

指定されたオプションを設定します。option に null が指定された場合は，デフォルトのオプションが指定されたものとします。オプションのデフォルト値については，「10.2.4　Option クラス」を参照してください。

戻り値

なし

例外

例外	内容
TranslatorException	パラメタの更新に失敗した場合に発生します。

10.2.2　MDLInfo クラス

ここでは，MDLInfo クラスの仕様について説明します。

（1）MDLInfo クラスの概要

継承

```
java.lang.Object
|
+-- jp.co.Hitachi.soft.interschema2.MDLInfo
```

形式

```
public final class MDLInfo
extends java.lang.Object
```

説明

トランスレータの動作を定義する MDL 情報を定義します。MDLInfo クラスのインスタンスは，Translator クラスの exec メソッドの引数として渡します。変換に使用中の MDLInfo クラスのインスタンスに対して，reload メソッドを呼び出すことはできません。

コンストラクタの概要

形式	機能
public MDLInfo(String mdlFileName)	指定された MDL ファイルをロードして，MDL 情報を生成します。

メソッドの概要

形式	機能
public DLProperty getDLProperty()	この MDL 情報の DL プロパティ情報を取得します。
public String getFileName()	この MDL 情報のファイル名を取得します。
public void reload(String mdlFileName)	指定されたファイル名で MDL 情報を更新します。

（2） コンストラクタの詳細

（a） MDLInfo

形式

```
public MDLInfo(String mdlFileName)
    throws TranslatorException
```

引数

引数	内容
mdlFileName	MDL ファイル名を指定します。

説明

指定された MDL ファイルをロードして，MDL 情報を生成します。mdlFileName に null 又は空文字列が指定された場合は，パラメタエラーとして TranslatorException が発生します。

例外

例外	内容
TranslatorException	MDL ファイルが未検証の場合，又は MDL ファイルが存在しない場合に発生します。

（3） メソッドの詳細

（a） getDLProperty

形式

```
public DLProperty getDLProperty()
    throws TranslatorException
```

引数

なし

説明

この MDL 情報の DL プロパティ情報を取得します。

戻り値

この MDL 情報の DL プロパティ情報が返されます。

例外

例外	内容
TranslatorException	DL プロパティ情報を構築できなかった場合に発生します。

(b) getFileName

形式

```
public String getFileName()
```

引数

なし

説明

この MDL 情報のファイル名を取得します。

戻り値

MDL 情報のファイル名が返されます。

(c) reload

形式

```
public void reload(String mdlFileName)
    throws TranslatorException
```

引数

引数	内容
mdlFileName	ロードする MDL ファイル名を指定します。

説明

指定のファイル名で MDL 情報を更新します。現在の MDL 情報を破棄して，指定の MDL ファイル名を再ロードします。

戻り値

なし

例外

例外	内容
TranslatorException	MDL ファイルが未検証の場合，又は MDL ファイルが存在しない場合に発生します。

10.2.3 DLProperty クラス

ここでは，DLProperty クラスの仕様について説明します。

（1） DLProperty クラスの概要

継承

```
java.lang.Object
|
+-- jp.co.Hitachi.soft.interschema2.DLProperty
```

形式

```
public final class DLProperty
extends java.lang.Object
```

説明

DL プロパティ情報を定義します。DL プロパティ情報には，次の情報が含まれます。

- ユーザバージョン
- ユーザコメント
- 作成日時
- 作成製品バージョン
- 更新日時
- 更新製品バージョン
- DL ファイルバージョン

ユーザバージョン及びユーザコメントには，FDL ファイル又は MDL ファイル作成時に設定された値を Unicode に変換した値が設定されます。FDL ファイル又は MDL ファイル作成時に設定された値がない場合は，null が設定されます。コード変換に失敗した場合は，DL プロパティ情報構築時に例外がスローされます。ただし，ワークステーションの OS で Interschema を使用する場合，LANG 環境変数に「C」が設定されているときは，FDL ファイル又は MDL ファイル作成時に設定された値を，JIS7（7bit JIS X0201（ISO646），カナなし）として扱います。
作成日時及び更新日時には，万国標準時（UCT）の 1970 年 1 月 1 日の 00:00:00 から，それぞれ作成日時又は更新日時までの通算経過ミリ秒が設定されます。
作成製品バージョン及び更新製品バージョンには，それぞれ作成及び更新に使用した Interschema のバージョンが，「VVRRSS」形式で設定されます。
DL ファイルバージョンには，FDL バージョン又は MDL バージョンが設定されま

す。
古いバージョンの FDL ファイル又は MDL ファイル（電子データ変換ツール又は Interschema バージョン 1 で作成したファイル）から作成された DL プロパティ情報の場合，有効な情報は，更新日時及び DL ファイルバージョンだけです。

コンストラクタの概要

形式	機能
public DLProperty(String dlFileName)	指定された FDL ファイル又は MDL ファイル名から，DL プロパティ情報を生成します。

メソッドの概要

形式	機能
public String getUserVersion()	この DL プロパティ情報のユーザバージョンを取得します。
public String getUserComment()	この DL プロパティ情報のユーザコメントを取得します。
public long getCreatedDate()	この DL プロパティ情報の作成日時を取得します。
public String getCreatedPPVersion()	この DL プロパティ情報の作成製品バージョンを取得します。
public long getLastUpdatedDate()	この DL プロパティ情報の更新日時を取得します。
public String getLastUpdatedPPVersion()	この DL プロパティ情報の更新製品バージョンを取得します。
public long getDLVersion()	この DL プロパティ情報の DL ファイルバージョンを取得します。

（2） コンストラクタの詳細

（a） DLProperty

形式

```
public DLProperty(String dlFileName)
    throws TranslatorException
```

引数

引数	内容
dlFileName	FDL ファイル名又は MDL ファイル名を指定します。

説明

指定された FDL ファイル又は MDL ファイルから，DL プロパティ情報を生成します。dlFileName に null 又は空文字列が指定された場合は，パラメタエラーとして TranslatorException が発生します。

例外

例外	内容
TranslatorException	不正なファイルが指定された場合，又はファイルが存在しなかった場合に発生します。

（3） メソッドの詳細

（a） getUserVersion

形式

```
public String getUserVersion()
```

引数

なし

説明

この DL プロパティ情報のユーザバージョンを取得します。

戻り値

ユーザバージョンが返されます。

（b） getUserComment

形式

```
public String getUserComment()
```

引数

なし

説明

この DL プロパティ情報のユーザコメントを取得します。

戻り値

ユーザコメントが返されます。

（c） getCreatedDate

形式

```
public long getCreatedDate()
```

引数

なし

説明

この DL プロパティ情報の作成日時を取得します。

戻り値

作成日時が戻されます。

（d） getCreatedPPVersion

形式

```
public String getCreatedPPVersion()
```

引数

なし

説明

この DL プロパティ情報の作成製品バージョンを取得します。

戻り値

作成製品バージョンが返されます。

（e） getLastUpdatedDate

形式

```
public long getLastUpdatedDate()
```

引数

なし

説明

この DL プロパティ情報の更新日時を取得します。

戻り値

更新日時が返されます。

（f） getLastUpdatedPPVersion

形式

```
public String getLastUpdatedPPVersion()
```

引数

なし

説明

この DL プロパティ情報の更新製品バージョンを取得します。

戻り値

更新製品バージョンが返されます。

（g）getDLVersion

形式

```
public long getDLVersion()
```

引数

なし

説明

この DL プロパティ情報の DL ファイルバージョンを取得します。

戻り値

DL ファイルバージョンが返されます。

10.2.4 Option クラス

ここでは，Option クラスの仕様について説明します。

（1）Option クラスの概要

継承

```
java.lang.Object
|
+-- jp.co.Hitachi.soft.interschema2.Option
```

形式

```
public final class Option
extends java.lang.Object
```

説明

トランスレータの変換オプションを定義します。変換オプションには，変換先ファイル出力方法，変換対象フォーマット指定，及び実行時オプションを設定します。このクラスのインスタンスを Translator クラスのインスタンスに設定して，変換時のオプションを指定することができます。

変換先ファイル出力方法には，新規作成（FA_UPDATE）及び追加書き（FA_APPEND）を設定することができます。デフォルトでは，新規作成（FA_UPDATE）が設定されます。

変換対象フォーマット指定には，指定なし（FN_OFF）又は指定あり（FN_ON）を設定することができます。デフォルトでは，指定なし（FN_OFF）が設定されます。

実行時オプションは，オプション種別及びそのオプションに対する引数値の組で構

成されます。指定したい実行時オプションを変換オプションに追加します。ただし，同じ実行時オプションを複数指定することはできません。同じ実行時オプションを複数回追加した場合は，最後に追加した値が有効となります。また，このクラスで指定した実行時オプションよりも，システム情報ファイル「ettrans.ini」で指定した実行時オプションが優先されます。システム情報ファイル「ettrans.ini」で指定する実行時オプションについては，「5.4.3　システム情報ファイル「ettrans.ini」での実行時オプションの定義」を参照してください。

フィールドの概要

形式	機能
public static final int FA_UPDATE	変換先ファイル出力方法の新規作成を表す定数です。
public static final int FA_APPEND	変換先ファイル出力方法の追加書きを表す定数です。
public static final boolean FN_OFF	変換対象フォーマット指定の指定なしを表す定数です。
public static final boolean FN_ON	変換対象フォーマット指定の指定ありを表す定数です。
public static final int OPT_CIIE	CIIE 実行時オプションを表す定数です。
public static final int OPT_CIIR	CIIR 実行時オプションを表す定数です。
public static final int OPT_CIIT	CIIT 実行時オプションを表す定数です。
public static final int OPT_CSV	CSV 実行時オプションを表す定数です。
public static final int OPT_DEFMAP	DEFMAP 実行時オプションを表す定数です。
public static final int OPT_DSEPA	DSEPA 実行時オプションを表す定数です。
public static final int OPT_DTM	DTM 実行時オプションを表す定数です。
public static final int OPT_EIAJHASH	EIAJHASH 実行時オプションを表す定数です。
public static final int OPT_ESEPA	ESEPA 実行時オプションを表す定数です。
public static final int OPT_FDS	FDS 実行時オプションを表す定数です。
public static final int OPT_IERR	IERR 実行時オプションを表す定数です。
public static final int OPT_SERR	SERR 実行時オプションを表す定数です。
public static final int OPT_SNCODE	SNCODE 実行時オプションを表す定数です。
public static final int OPT_XDOC	XDOC 実行時オプションを表す定数です。
public static final int OPT_XND	XND 実行時オプションを表す定数です。
public static final int OPT_XVC	XVC 実行時オプションを表す定数です。
public static final int OPT_XWS	XWS 実行時オプションを表す定数です。

コンストラクタの概要

形式	機能
public Option()	デフォルトの変換オプションを生成します。

メソッドの概要

形式	機能
public boolean addOption(int optionKind, String value)	指定された実行時オプション及び引数を追加します。
public boolean addOption(int optionKind)	指定された実行時オプションを追加します。
public boolean removeOption(int optionKind)	指定された実行時オプションを削除します。
public int getFileAppendFlag()	現在の変換先ファイル出力方法を取得します。
public void setFileAppendFlag(int FileAppendFlag)	変換先ファイル出力方法を設定します。
public boolean getFNFlag()	現在の変換対象フォーマット指定を取得します。
public void setFNFlag(boolean FNFlag)	変換対象フォーマット指定を設定する。
public String getOptionString()	現在の実行時オプションの文字列を取得します。
public boolean containsOption(int optionKind)	実行時オプションが指定されているかを判定します。

（2） フィールドの詳細

（a） FA_UPDATE

形式

```
public static final int FA_UPDATE = 0
```

説明

変換先ファイル出力方法の新規作成を表す定数です（デフォルト）。変換先ファイルを出力データで上書きします。ファイルがない場合は新規に作成します。

（b） FA_APPEND

形式

```
public static final int FA_APPEND = 1
```

説明

変換先ファイル出力方法の追加書きを表す定数です。出力データを変換先ファイルの最後に追加します。ファイルがない場合は新規に作成します。

（c） FN_OFF

形式

```
public static final boolean FN_OFF = false
```

説明

変換対象フォーマット指定の指定なしを表す定数です（デフォルト）。MDL ファイル内のすべてのフォーマットを変換します。

(d) FN_ON

形式

```
public static final boolean FN_ON = true
```

説明

変換対象フォーマット指定の指定なしを表す定数です。MDL ファイル内のパラメタで指定されたフォーマットだけを変換します。

(e) OPT_CIIE

形式

```
public static final int OPT_CIIE = 10000
```

説明

CIIE 実行時オプションを表す定数です。実行時オプションの詳細は，「5.4.1(3) 引数」を参照してください。

(f) OPT_CIIR

形式

```
public static final int OPT_CIIR = 10001
```

説明

CIIR 実行時オプションを表す定数です。実行時オプションの詳細は，「5.4.1(3) 引数」を参照してください。

(g) OPT_CIIT

形式

```
public static final int OPT_CIIT = 10002
```

説明

CIIT 実行時オプションを表す定数です。実行時オプションの詳細は，「5.4.1(3) 引数」を参照してください。

(h) OPT_CSV

形式

```
public static final int OPT_CSV = 10003
```

説明

CSV 実行時オプションを表す定数です。実行時オプションの詳細は，「5.4.1(3) 引数」を参照してください。

(i) OPT_DEFMAP

形式

```
public static final int OPT_DEFMAP = 10004
```

説明

DEFMAP 実行時オプションを表す定数です。実行時オプションの詳細は，「5.4.1(3) 引数」を参照してください。

(j) OPT_DSEPA

形式

```
public static final int OPT_DSEPA = 10005
```

説明

DSEPA 実行時オプションを表す定数です。実行時オプションの詳細は，「5.4.1(3) 引数」を参照してください。

(k) OPT_DTM

形式

```
public static final int OPT_DTM = 10006
```

説明

DTM 実行時オプションを表す定数です。実行時オプションの詳細は，「5.4.1(3) 引数」を参照してください。

(l) OPT_EIAJHASH

形式

```
public static final int OPT_EIAJHASH = 10007
```

説明

EIAJHASH 実行時オプションを表す定数です。実行時オプションの詳細は，「5.4.1(3) 引数」を参照してください。

（m）OPT_ESEPA

形式

```
public static final int OPT_ESEPA = 10008
```

説明

ESEPA 実行時オプションを表す定数です。実行時オプションの詳細は，「5.4.1(3) 引数」を参照してください。

（n）OPT_FDS

形式

```
public static final int OPT_FDS = 10015
```

説明

FDS 実行時オプションを表す定数です。実行時オプションの詳細は，「5.4.1(3) 引数」を参照してください。

（o）OPT_IERR

形式

```
public static final int OPT_IERR = 10009
```

説明

IERR 実行時オプションを表す定数です。実行時オプションの詳細は，「5.4.1(3) 引数」を参照してください。

（p）OPT_SERR

形式

```
public static final int OPT_SERR = 10010
```

説明

SERR 実行時オプションを表す定数です。実行時オプションの詳細は，「5.4.1(3) 引数」を参照してください。

（q）OPT_SNCODE

形式

```
public static final int OPT_SNCODE = 10017
```

説明

SNCODE 実行時オプションを表す定数です。実行時オプションの詳細は，「5.4.1(3) 引数」を参照してください。

(r) OPT_XDOC

形式

```
public static final int OPT_XDOC = 10011
```

説明

XDOC 実行時オプションを表す定数です。実行時オプションの詳細は，「5.4.1(3) 引数」を参照してください。

(s) OPT_XND

形式

```
public static final int OPT_XND = 10016
```

説明

XND 実行時オプションを表す定数です。実行時オプションの詳細は，「5.4.1(3) 引数」を参照してください。

(t) OPT_XVC

形式

```
public static final int OPT_XVC = 10012
```

説明

XVC 実行時オプションを表す定数です。実行時オプションの詳細は，「5.4.1(3) 引数」を参照してください。

(u) OPT_XWS

形式

```
public static final int OPT_XWS = 10013
```

説明

XWS 実行時オプションを表す定数です。実行時オプションの詳細は，「5.4.1(3) 引数」を参照してください。

（3） コンストラクタの詳細

（a） Option

形式

```
public Option()
```

引数

なし

説明

デフォルトの変換オプションを生成します。
変換先ファイル出力方法には，新規作成（FA_UPDATE）が設定されます。
変換対象フォーマット指定には，指定なし（FN_OFF）が設定されます。
実行時オプションには，指定なしが設定されます。

（4） メソッドの詳細

（a） addOption

形式

```
public boolean addOption(int optonKind, String value)
```

引数

引数	内容
optionKind	実行時オプション種別を指定します。
value	実行時オプション引数を指定します。

説明

指定された実行時オプション，及びその引数を追加します。
追加済み実行時オプションを指定した場合は，指定の値で古い値を置き換えます。
引数値が必要ないオプションに対して引数値を設定した場合，その値は無視されます。

戻り値

指定された実行時オプションを追加できる場合は true が返されます。追加できない場合は false が返されます。

（b） addOption

形式

```
public boolean addOption(int optonKind)
```

引数

引数	内容
optionKind	実行時オプション種別を指定します。

説明

指定の実行時オプションを追加します。追加済みの実行時オプションを指定した場合は，指定の値で古い値を置き換えます。

戻り値

指定された実行時オプションを追加できる場合は true が返されます。追加できない場合は false が返されます。

（c） removeOption

形式

```
public boolean removeOption(int optionKind)
```

引数

引数	内容
optionKind	実行時オプション種別を指定します。

説明

指定された実行時オプションを削除します。

戻り値

指定された実行時オプションを削除できる場合は true が返されます。追加できない場合は false が返されます。

（d） getFileAppendFlag

形式

```
public int getFileAppendFlag()
```

引数

なし

説明

現在の変換先ファイル出力方法を取得します。

戻り値

変換先出力ファイル出力方法が返されます。

（e） setFileAppendFlag

形式

```
public void setFileAppendFlag(int FileAppendFlag)
```

引数

引数	内容
FileAppendFlag	変換先ファイル出力方法を指定します。

説明

変換先ファイル出力方法を設定します。新規作成（FA_UPDATE）又は追加書き（FA_APPEND）を設定します。これら以外の値を設定した場合は，デフォルトで新規作成（FA_UPDATE）を設定します。

戻り値

なし

（f） getFNFlag

形式

```
public boolean getFNFlag()
```

引数

なし

説明

現在の変換対象フォーマット指定を取得します。

戻り値

変換対象フォーマット指定が返されます。

（g） setFNFlag

形式

```
public void setFNFlag(boolean FNFlag)
```

引数

引数	内容
FNFlag	変換対象フォーマット指定を指定します。

説明

変換対象フォーマット指定を設定します。指定なし（FN_OFF）又は指定あり

（FN_ON）を設定します。

戻り値

なし

（h） getOptionString

形式

```
public String getOptionString()
```

引数

なし

説明

現在の実行時オプションの文字列を取得します。

戻り値

実行時オプションの文字列を返します。

（i） containsOption

形式

```
public boolean containsOption(int optionKind)
```

引数

引数	内容
optionKind	判定対象となる実行時オプションを表す定数を指定します。

説明

実行時オプションが指定されているかどうかを判定します。

戻り値

実行時オプションが指定されている場合は true が返されます。指定されていない場合は false が返されます。

10.2.5 TranslateData クラス

ここでは，TranslateData クラスの仕様について説明します。

（1） TranslateData クラスの概要

継承

```
java.lang.Object
|
+-- jp.co.Hitachi.soft.interschema2.TranslateData
```

直系の既知のサブクラス

FileData，StringData，InputStreamData，OutputStreamData

形式

```
public abstract class TranslateData
extends java.lang.Object
```

説明

変換対象データの抽象クラスを定義します。変換対象データは，フォーマット名，フォーマット名に対応するデータオブジェクト，データエリア初期サイズ，及びデータエリア増分サイズを持ちます。Translator クラスの exec メソッドを実行する際に，このクラスを指定して変換を行います。

フォーマット名には，変換対象となる FDL のフォーマット名又は DTD のルート要素名が指定されます。

データオブジェクトには，フォーマット名に対応する固有のデータを表現した Java オブジェクトが指定されます。

データエリア初期サイズ及びデータエリア増分サイズは，変換時に確保するデータエリアのサイズを指定する場合に設定します。

これらの値は，変換対象データが出力データとして指定された場合だけ有効です。

ユーザは，このクラスを拡張して独自の変換対象データを定義することで，任意の形式のデータオブジェクトを変換することができます。ユーザ独自の変換対象データの定義については，「10.4(2)　サンプルコード（TranslateData 派生クラス）」を参照してください。

また，toByteArray メソッド及び toDataObject メソッドを実装する場合，エラーを報告するときは，必ず UserException 又はその派生クラスの例外をスローしてください。それ以外の例外をスローしたときは，Interschema は予期しないエラーとして処理します。

メソッドの概要

形式	機能
public final String getFormatName()	現在のフォーマット名を取得します。
public final void setFormatName(String formatName)	指定されたフォーマット名を設定します。
public final Object getDataObject()	現在のデータオブジェクトを取得します。
public final void setDataObject(Object data)	指定されたデータオブジェクトを設定します。
public final int getMemoryInitSize()	現在の出力データエリア初期サイズを取得します。

形式	機能
public inal void setMemoryInitSize(int initSize)	指定された出力データエリア初期サイズを設定します。
public final init getMemoryIncrementSize()	現在の出力データエリア増分サイズを取得します。
public final void setMemoryIncrementSize(int incrementSize)	指定のされた出力データエリア増分サイズを設定します。
protected abstract byte[] toByteArray(Object obj)	指定されたデータオブジェクトをバイト列に変換します。
protected abstract Object toDataObject(byte[] byteData)	指定されたバイト列をデータオブジェクトに変換します。

（2） メソッドの詳細

（a） getFormatName

形式

```
public final String getFormatName()
```

引数

なし

説明

現在のフォーマット名を取得します。

戻り値

フォーマット名が返されます。

（b） setFormatName

形式

```
public final void setFormatName(String formatName)
```

引数

引数	内容
formatName	フォーマット名を指定します。

説明

指定されたフォーマット名を設定します。

戻り値

なし

（c） getDataObject

形式

```
public final Object getDataObject()
```

引数

なし

説明

現在のデータオブジェクトを取得します。

戻り値

変換対象フォーマットデータを返します。

（d） setDataObject

形式

```
public final void setDataObject(Object data)
```

引数

引数	内容
data	データオブジェクトを指定します。

説明

指定されたデータオブジェクトを設定します。

戻り値

なし

（e） getMemoryInitSize

形式

```
public final int getMemoryInitSize()
```

引数

なし

説明

現在の出力データエリア初期サイズを取得します。

戻り値

出力データエリア初期サイズが返されます。

(f) setMemoryInitSize

形式

```
public final void setMemoryInitSize(int initSize)
```

引数

引数	内容
initSize	出力データエリア初期サイズを指定します。

説明

指定された出力データエリア初期サイズを設定します。
この値に 0 が設定された場合は，システム情報ファイル「ettrans.ini」内の [MemoryData] セクションで定義された値を使用します。システム情報ファイル「ettrans.ini」に定義がない場合は，1024 バイトを仮定します。

戻り値

なし

(g) getMemoryIncrementSize

形式

```
public final int getMemoryIncrementSize()
```

引数

なし

説明

現在の出力データエリア増分サイズを取得します。

戻り値

出力データエリア増分サイズを指定します。

(h) setMemoryIncrementSize

形式

```
public final void setMemoryIncrementSize(int incrementSize)
```

引数

引数	内容
incrementSize	出力データエリア増分サイズを指定します。

説明

指定された出力データエリア増分サイズを設定します。
この値に 0 が設定された場合は，システム情報ファイル「ettrans.ini」内の [MemoryData] セクションで定義された値を使用します。システム情報ファイル「ettrans.ini」に定義がない場合は，1024 バイトを仮定します。

戻り値

なし

(i) toByteArray

形式

```
protected abstract byte[] toByteArray(Object obj)
    throws TranslatorException, UserException
```

引数

引数	内容
obj	データオブジェクトを指定します。

説明

指定されたデータオブジェクトをバイト列に変換します。このメソッドは抽象メソッドであるため，サブクラスは必ずこのメソッドを実装する必要があります。サブクラス固有のデータオブジェクトをバイト列に変換する方法を実装しています。
通常は，「任意のデータオブジェクト x について，
x.equals(toDataObject(toByteArray(x)) が true を返す」という要件を満たすように実装します。

戻り値

データオブジェクトのバイト列が返されます。

例外

例外	内容
TranslatorException	データオブジェクトのバイト列への変換に失敗した場合に発生します。
UserException	ユーザが定義した変換対象データで例外が発生した場合に発生します。

(j) toDataObject

形式

```
protected abstract Object toDataObject ( byte[] byteData )
    throws TranslatorException, UserException
```

引数

引数	内容
byteData	データオブジェクトのバイト列を指定します。

説明

指定されたバイト列をデータオブジェクトに変換します。このメソッドは抽象メソッドであるため，サブクラスは必ずこのメソッドを実装する必要があります。バイト列をサブクラス固有のデータオブジェクトに変換する方法を実装します。通常は，「任意の変換対象フォーマットデータ x について，x.equals(toDataObject(toByteArray(x)) が true を返す」という要件を満たすように実装します。

戻り値

データオブジェクトが返されます。

例外

例外	内容
TranslatorException	バイト列のデータオブジェクトへの変換に失敗した場合に発生します。
UserException	ユーザが定義した変換対象データで例外が発生した場合に発生します。

10.2.6 FileData クラス

ここでは，FileData クラスの仕様について説明します。

（1） FileData クラスの概要

継承

```
java.lang.Object
|
+-- jp.co.Hitachi.soft.interschema2.TranslateData
    |
    +-- jp.co.Hitachi.soft.interschema2.FileData
```

形式

```
public final class FileData
extends jp.co.Hitachi.soft.interschema2.TranslateData
```

説明

ファイル形式の変換対象データを定義します。ファイル形式のデータオブジェクトを変換する場合に使用します。この変換対象データは，入力及び出力のどちらにも使用できます。

コンストラクタの概要

形式	機能
public FileData(String formatName, File file)	指定されたフォーマット名及びファイルから，ファイル形式の変換対象データを生成します。
public FileData(String formatName, String fileName)	指定されたフォーマット名及びファイル名から，ファイル形式の変換対象データを生成します。

メソッドの概要

形式	機能
public String getFileName()	現在の変換対象データのファイル名を取得します。
protected byte[] toByteArray(Object obj)	指定されたデータオブジェクトをバイト列に変換します。
protected Object toDataObject(byte[] byteData)	指定されたバイト列をデータオブジェクトに変換します。

（2）コンストラクタの詳細

（a）FileData

形式

```
public FileData(String formatName, File file)
```

引数

引数	内容
formatName	フォーマット名を指定します。
file	ファイルを指定します。

説明

指定されたフォーマット名及びファイルから，ファイル形式の変換対象データを生成します。

（b）FileData

形式

```
public FileData(String formatName, String fileName)
```

引数

引数	内容
formatName	フォーマット名を指定します。
fileName	ファイル名を指定します。

説明

指定されたフォーマット名及びファイル名から，ファイル形式の変換対象データを生成します。

（3） メソッドの詳細

（a） getFileName

形式

```
public String getFileName()
```

引数

なし

説明

現在のデータオブジェクトのファイル名を取得します。

戻り値

ファイル形式データオブジェクトのファイル名が返されます。

（b） toByteArray

形式

```
protected byte[] toByteArray (Object obj)
    throws TranslatorException
```

引数

引数	内容
obj	ファイル形式のデータオブジェクトを指定します。

説明

指定されたデータオブジェクトをバイト列に変換します。ファイル形式のデータオブジェクトを読み込みバイト列に変換します。

戻り値

データオブジェクトのバイト列が返されます。

例外

例外	内容
TranslatorException	指定されたデータオブジェクトが File クラス，又は String クラスのインスタンスでない場合，又はデータオブジェクトのバイト列への変換に失敗した場合に発生します。

（c） toDataObject

形式

```
protected Object toDataObject(byte[] byteData)
    throws TranslatorException
```

引数

引数	内容
byteData	データオブジェクトのバイト列を指定します。

説明

指定されたバイト列をデータオブジェクトに変換します。指定されたバイト列を，現在のデータオブジェクトに設定されているファイルに書き込んで，現在のデータオブジェクトを返します。

戻り値

ファイル形式のデータオブジェクトを返します。

例外

例外	内容
TranslatorException	バイト列の変換対象フォーマットデータへの変換に失敗した場合に発生します。

10.2.7 StringData クラス

ここでは，StringData クラスの仕様について説明します。

（1） StringData クラスの概要

継承

```
java.lang.Object
|
+-- jp.co.Hitachi.soft.interschema2.TranslateData
    |
    +-- jp.co.Hitachi.soft.interschema2.StringData
```

形式

```
public final class StringData
extends jp.co.Hitachi.soft.interschema2.TranslateData
```

説明

文字列形式の変換対象データを定義します。Java 言語の文字列形式のデータオブジェクトを変換する場合に使用します。この変換対象データは，入力及び出力のどちらにも使用することができます。

コンストラクタの概要

形式	機能
public StringData(String formatName, String data)	指定されたフォーマット名及び文字列データから，文字列形式の変換対象データを生成します。

メソッドの概要

形式	機能
protected byte[] toByteArray(Object obj)	指定されたデータオブジェクトをバイト列に変換します。
protected Object toDataObject(byte[] byteData)	指定されたバイト列をデータオブジェクトに変換します。

(2) コンストラクタの詳細

(a) StringData

形式

```
public StringData(String formatName, String data)
```

引数

引数	内容
formatName	フォーマット名を指定します。
data	文字列形式のデータオブジェクトを指定します。

説明

指定されたフォーマット名及び文字列データから文字列形式の変換対象データを生成します。

(3) メソッドの詳細

(a) toByteArray

形式

```
protected byte[] toByteArray (Object obj)
```

```
    throws TranslatorException
```

引数

引数	内容
obj	文字列形式のデータオブジェクトを指定します。

説明

指定されたデータオブジェクトをバイト列に変換します。指定の文字列形式のデータオブジェクトを，プラットフォームデフォルトの文字列セットを使用してバイト列に変換します。

戻り値

データオブジェクトのバイト列が返されます。

例外

例外	内容
TranslatorException	指定されたデータオブジェクトが String クラスのインスタンスでない場合，又はデータオブジェクトをバイト列に変換失敗した場合に発生します。

(b) toDataObject

形式

```
protected Object toDataObject (byte[] byteData)
    throws TranslatorException
```

引数

引数	内容
byteData	データオブジェクトのバイト列を指定します。

説明

指定されたバイト列をデータオブジェクトに変換します。指定されたバイト列をプラットフォームデフォルトの文字セットを使用して文字列形式のデータオブジェクトに変換します。

戻り値

文字列形式のデータオブジェクトが返されます。

例外

例外	内容
TranslatorException	バイト列のデータオブジェクトへの変換に失敗した場合に発生します。

10.2.8 InputStreamData クラス

ここでは，InputStreamData クラスの仕様について説明します。

（1） InputStreamData クラスの概要

継承

```
java.lang.Object
|
+-- jp.co.Hitachi.soft.interschema2.TranslateData
    |
    +-- jp.co.Hitachi.soft.interschema2.InputStreamData
```

形式

```
public final class InputStreamData
extends jp.co.Hitachi.soft.interschema2.TranslateData
```

説明

入力ストリーム形式の変換対象データを定義します。Java 言語の入力ストリーム形式のデータオブジェクトを変換する場合に使用します。この変換対象データは入力専用です。このクラスのインスタンスを Translator クラスの exec メソッドの出力データとして指定した場合は，TranslatorException が発生します。

exec メソッド実行後は，このクラスに指定した InputStream のインスタンスは更新されます。

コンストラクタの概要

形式	機能
public InputStreamData(String formatName, InputStream streamData)	指定のフォーマット名及び入力ストリームから，入力ストリーム形式の変換対象データを生成します。

メソッドの概要

形式	機能
protected byte[] toByteArray(Object obj)	指定のデータオブジェクトをバイト列に変換します。
protected Object toDataObject(byte[] byteData)	指定のバイト列をデータオブジェクトに変換します。

（2） コンストラクタの詳細

（a） InputStreamData

形式

```
public InputStreamData(String formatName, InputStream
```

```
streamData)
```

引数

引数	内容
formatName	フォーマット名を指定します。
streamData	入力ストリーム形式のデータオブジェクトを指定します。

説明

指定されたフォーマット名及び入力ストリームから，入力ストリーム形式の変換対象データを生成します。変換データは指定の入力ストリームから読み込まれます。

（3） メソッドの詳細

（a） toByteArray

形式

```
protected byte[] toByteArray(Object obj)
    throws TranslatorException
```

引数

引数	内容
obj	入力ストリーム形式のデータオブジェクトを指定します。

説明

指定されたデータオブジェクトをバイト列に変換します。指定の入力ストリーム形式のデータオブジェクトからデータを読み取って，バイト列に変換します。

戻り値

データオブジェクトから読み込んだバイト列が返されます。

例外

例外	内容
TranslatorException	指定されたデータオブジェクトが InputStream のインスタンスでない場合，又はデータオブジェクトのバイト列への変換に失敗した場合に発生します。

（b） toDataObject

形式

```
protected Object toDataObject(byte[] byteData)
    throws TranslatorException
```

引数

引数	内容
byteData	データオブジェクトのバイト列を指定します。

説明

指定されたバイト列をデータオブジェクトに変換します。InputStreamData クラスは入力専用の変換対象データであるため，このメソッドをサポートしません。このメソッドを実行すると TranslatorException が発生します。

戻り値

データオブジェクトが返されます。

例外

例外	内容
TranslatorException	必ず発生します。

10.2.9 OutputStreamData クラス

ここでは，OutputStreamData クラスの仕様について説明します。

（1） OutputStreamData クラスの概要

継承

```
java.lang.Object
|
+-- jp.co.Hitachi.soft.interschema2.TranslateData
    |
    +-- jp.co.Hitachi.soft.interschema2.OutputStreamData
```

形式

```
public final class OutputStreamData
extends jp.co.Hitachi.soft.interschema2.TranslateData
```

説明

出力ストリーム形式の変換対象データを定義します。Java 言語の出力ストリーム形式のデータオブジェクトを変換する場合に使用します。この変換対象データは出力専用です。このクラスのインスタンスを Translator クラスの exec メソッドの入力データとして指定した場合は，TranslatorException が発生します。

コンストラクタの概要

形式	機能
public OutputStreamData(String formatName, OutputStream streamData)	指定のされたフォーマット名及び出力ストリームから，出力ストリーム形式の変換対象データを生成します。

メソッドの概要

形式	機能
protected byte[] toByteArray(Object obj)	指定されたデータオブジェクトをバイト列に変換します。
protected Object toDataObject(byte[] byteData)	指定されたバイト列をデータオブジェクトに変換します。

（2） コンストラクタの詳細

（a） OutputStreamData

形式

```
public OutputStreamData(String formatName, OutputStream
streamData)
```

引数

引数	内容
formatName	フォーマット名を指定します。
streamData	出力ストリーム形式のデータオブジェクトを指定します。

説明

指定されたフォーマット名及び出力ストリームから，出力ストリーム形式の変換対象データを生成します。変換後のデータは指定の出力ストリームに出力されます。

（3） メソッドの詳細

（a） toByteArray

形式

```
protected byte[] toByteArray(Object obj)
    throws TraslatorException
```

引数

引数	内容
obj	データオブジェクトを指定します。

説明

指定されたデータオブジェクトをバイト列に変換します。OutputStreamData クラスは出力専用の変換対象データのため，このメソッドをサポートしません。このメソッドを実行すると，TranslatorException が発生します。

戻り値

データオブジェクトのバイト列が返されます。

例外

例外	内容
TranslatorException	必ず発生します。

（b） toDataObject

形式

```
protected Object toDataObject(byte[] byteData)
    throws TraslatorException
```

引数

引数	内容
byteData	データオブジェクトのバイト列を指定します。

説明

指定されたバイト列をデータオブジェクトに変換します。指定されたバイト列を現在のデータオブジェクトの出力ストリームに書き込んで，現在のデータオブジェクトを返します。

戻り値

出力ストリーム形式のデータオブジェクトが返されます。

例外

例外	内容
TranslatorException	バイト列のデータオブジェクトへの変換に失敗した場合に発生します。

10.2.10 TranslatorException クラス

ここでは，TranslatorException クラスの仕様について説明します。

（1） TranslatorException クラスの概要

継承

```
java.lang.Object
```

```
|
+--java.lang.Throwable
   |
   +--java.lang.Exception
      |
      +-- jp.co.Hitachi.soft.interschema2.TranslatorException
```

形式

```
public final class TranslatorException
extends java.lang.Exception
```

説明

データ変換処理 API（Java 言語）実行時に発生したエラー例外を定義します。getErrorCode メソッドで取得するエラーコードは，データ変換処理 API（C 言語）及び ettrans コマンドのエラーコードとは異なります。ただし，幾つかのエラーには，データ変換処理 API（C 言語）及び ettrans コマンドと共通のエラー情報を取得できるものがあります。共通のエラー情報は，getExtraErrorData メソッドを使用して取得します。取得した値については，「10.2.11　ExtraErrorData クラス」を参照してください。拡張エラー情報は，Translator クラスの exec メソッド呼び出し以外では設定されません。

メソッドの概要

形式	機能
public int getErrorCode()	データ変換処理 API（C 言語）で定義したエラーコードを取得します。
public String getMessage()	例外の発生要因となったエラーの内容を示す文字列を取得します。
public ExtraErrorData[] getExtraErrorData()	拡張エラー情報を取得します。

（2） メソッドの詳細

（a） getErrorCode

形式

```
public int getErrorCode()
```

引数

なし

説明

エラーコードを取得します。エラーコードの詳細は，「付録 A.1(3)(b)　エラーコード」を参照してください。

戻り値

エラーコードが返されます。

（b） getMessage

形式

```
public String getMessage()
```

引数

なし

説明

例外の詳細メッセージを取得します。

戻り値

エラー内容を示すメッセージが返されます。

（c） getExtraErrorData

形式

```
public ExtraErrorData[] getExtraErrorData()
```

引数

なし

説明

拡張エラー情報を取得します。ExtraErrorData の詳細は「10.2.11 ExtraErrorData クラス」を参照してください。

戻り値

拡張エラー情報の配列が返されます。

10.2.11 ExtraErrorData クラス

ここでは，ExtraErrorData クラスの仕様について説明します。

（1） ExtraErrorData クラスの概要

継承

```
java.lang.Object
|
+-- jp.co.Hitachi.soft.interschema2.ExtraErrorData
```

形式

```
public final class ExtraErrorData
```

```
extends java.lang.Object implements java.io.Serializable
```

説明

変換実行時に発生した拡張エラー情報を定義します。TranslatorException 発生時に詳細なエラー情報がある場合に設定されます。設定される値は，データ変換処理 API（C 言語）のエラー情報と同じです。値の意味は，「9.2.9　ETtrans2Exec（変換実行）」の「表 9-2　ログファイルに出力される数値情報，文字情報の内容」を参照してください。

メソッドの概要

形式	機能
public int getMessageNo()	メッセージ番号を取得します。
public int getErrorLevel()	エラーレベルを取得します。
public int getNumericData()	数値情報を取得します。
public String getInformation()	エラーの内容を示す文字列情報を取得します。

（2）メソッドの詳細

（a）getMessageNo

形式

```
public int getMessageNo()
```

引数

なし

説明

メッセージ番号を取得します。

戻り値

メッセージ番号が返されます。

（b）getErrorLevel

形式

```
public int getErrorLevel()
```

引数

なし

説明

エラーレベルを取得します。

戻り値
エラーレベルが返されます。

（c） getNumericData

形式
```
public int getNumericData()
```

引数
なし

説明
数値情報を取得します。

戻り値
数値情報が返されます。

（d） getInformation

形式
```
public String getInformation()
```

引数
なし

説明
エラーの内容を示す文字列情報を取得します。

戻り値
エラーの内容を示す文字列情報が返されます。

10.2.12 UserException クラス

ここでは，UserException クラスの仕様について説明します。

（1） UserException クラスの概要

継承
```
java.lang.Object
|
+--java.lang.Throwable
   |
   +--java.lang.Exception
      |
      +-- jp.co.Hitachi.soft.interschema2.UserException
```

直系の既知のサブクラス

ExitFuncWarning，ExitFuncException

形式

```
public class UserException
extends java.lang.Exception
```

説明

ユーザ作成クラスで発生したエラー例外を定義します。
ユーザが独自に作成したクラスで例外を発生させる場合は，このクラス又はこのクラスのサブクラスを定義してエラーを報告します。
ユーザ定義クラス内でこのクラス，又はこのクラスの派生クラスの例外が発生した場合は，そのまま上位へスローします。
ユーザ定義クラスでこのクラス，又はこのクラスの派生クラス以外の例外が発生した場合は，TranslatorException が発生します。

コンストラクタの概要

形式	機能
public UserException(String message)	指定された詳細メッセージを持つユーザ例外を生成します。
public UserException()	デフォルトのユーザ例外を生成します。

（2）コンストラクタの詳細

（a）UserException

形式

```
public UserException(String message)
```

引数

引数	内容
message	詳細メッセージを指定します。

説明

指定された詳細メッセージを持つユーザ例外を生成します。

（b）UserException

形式

```
public UserException()
```

引数

なし

説明

デフォルトのユーザ例外を生成します。詳細メッセージに null を指定して，ユーザ例外を生成する場合と等価です。

10.3　jp.co.Hitachi.soft.interschema2.exitfunc パッケージのクラスの仕様

この節では，jp.co.Hitachi.soft.interschema2.exitfunc パッケージに含まれる Java クラスの仕様について説明します。この節の記述形式を次に示します。

継承

Java クラスの継承関係を示します。

形式

Java クラス，フィールド，コンストラクタ，及びメソッドの記述形式を示します。

引数

Java クラス，コンストラクタ，及びメソッドの引数を示します。

説明

Java クラス，フィールド，コンストラクタ，及びメソッドの機能について説明します。

戻り値

Java クラス及びメソッドの戻り値を示します。

例外

Java クラス，コンストラクタ，及びメソッドの例外を示します。また，例外が発生する条件について説明します。

10.3.1　ENVString クラス

ここでは，ENVString クラスの仕様について説明します。

（1）ENVString クラスの概要

継承

```
java.lang.Object
|
+-- jp.co.Hitachi.interschema2.exitfunc.ENVString
```

形式

```
public final class ENVString
extends java.lang.Object
```

説明

Interschema の ET_ENVSTRING 型に対応する Java クラスです。出口関数の引数として使用します。Interschema が出口関数の引数に文字列を渡す際に文字列データを String に変換した文字列，及び変換時のエラーコードを持ちます。変換結果とエラーコードに格納される値の関係を，次の表に示します。

表 10-3　変換結果とエラーコードに格納される値の関係

エラーコード	文字コード変換結果	文字列
0	変換に成功しました。	String 文字列
0x01000001	変換時に不正な文字コードを検出しました。※	不正な文字コードをスペースに置換した String 文字列
-1	変換に失敗しました。	null

注※

フォーマット指定又は実行時オプションで不正文字コードのスペース置換を禁止している場合，変換時に不正文字コードを検出したときは，変換失敗として処理します。

メソッドの概要

形式	機能
public String getString()	文字列を取得します。
public int getErrorCode()	文字コード変換のエラーコードを取得します。

(2) メソッドの詳細

(a) getString

形式

```
public String getString()
```

引数

なし

説明

文字列を取得します。

戻り値

文字列が返されます。

(b) getErrorCode

形式

```
public int getErrorCode()
```

引数

なし

説明

文字列変換時のエラーコードを取得します。

戻り値

文字列変換のエラーコードが返されます。

10.3.2 DateTime クラス

ここでは，DateTime クラスの仕様について説明します。

（1） DateTime クラスの概要

継承

```
java.lang.Object
|
+-- jp.co.Hitachi.interschema2.exitfunc.DateTime
```

形式

```
public final class DateTime
extends java.lang.Object
```

説明

Interschema の ET_DTM 型に対応する Java クラスです。出口関数の引数及び戻り値に使用します。Interschema から引数として受け取る場合，日付時刻部分の値が未設定のときは 0 が設定されます。ゾーン文字列部分の値が未設定のときは null が設定されます。

コンストラクタの概要

形式	機能
public DateTime(int year, int month, int day, int hour, int minute, double second, String timeZone)	指定の値を持つ日付時刻を生成します。
public DateTime(Date date, String timeZone)	指定の値を持つ日付時刻を生成します。
public DateTime(Date date)	指定の値を持つ日付時刻を生成します。
public DateTime()	デフォルトの日付時刻を生成します。

メソッドの概要

形式	機能
public int getYear()	年を取得します。

形式	機能
public void setYear(int year)	年を設定します。
public int getMonth()	月を取得します。
public void setMonth(int month)	月を設定します。
public int getDay()	日付を取得します。
public void setDay(int day)	日付を設定します。
public int getHour()	時を取得します。
public void setHour(int hour)	時を設定します。
public int getMinute()	分を取得します。
public void setMinute(int minute)	分を設定します。
public double getSecond()	秒を取得します。
public void setSecond(double second)	秒を設定します。
public String getZone()	タイムゾーン文字列を取得します。
public void setZone(String timeZone)	タイムゾーン文字列を設定します。
public Date getDate()	設定された時刻に対応する Date オブジェクトを取得します

（2） コンストラクタの詳細

（a） DateTime

形式

```
public DateTime(int year,
                int month,
                int day,
                int hour,
                int minute,
                double second,
                String timeZone)
    throws ExitFuncException
```

引数

引数	内容
year	年を指定します。
month	月を指定します。
hour	時を指定します。
day	日を指定します。
minute	分を指定します。

引数	内容
second	秒を指定します。
timeZone	タイムゾーン文字列を指定します。

説明

指定の値を持つ日付時刻を生成します。各引数の有効な範囲を次に示します。

- year(0 ～ 9999)
- month(1 ～ 12)
- day (1 ～ 31)
- hour(0 ～ 23)
- minute(0 ～ 59)
- second(0 ～ 59, 小数部可)

例外

例外	内容
ExitFuncException	引数に範囲外の値を指定した場合に発生します。

(b) DateTime

形式

```
public DateTime(Date date, String timeZone)
    throws ExitFuncException
```

引数

引数	内容
date	日付・時刻を指定します。
timeZone	タイムゾーン文字列です。

説明

指定された値を持つ日付時刻を生成します。年月日分秒に指定の日付・時刻を，タイムゾーン文字列に zone を指定して，日付時刻を生成する場合と等価です。

例外

例外	内容
ExitFuncException	date に null を指定した場合，又は日付・時刻の設定に失敗した場合に発生します。

(c) DateTime

形式

```
public DateTime(Date date)
    throws ExitFuncException
```

引数

引数	内容
date	日付・時刻を指定します。

説明

指定された値を持つ日付時刻を生成します。年月日分秒に指定の日付・時刻を，タイムゾーン文字列に現在のこのホストのデフォルトのタイムゾーン文字列（TimeZone.getDefault().getDisplayName() の値）を指定して，日付時刻を生成する場合と等価です。

例外

例外	内容
ExitFuncException	date に null を指定した場合，又は日付・時刻の設定に失敗した場合に発生します。

（d） DateTime

形式

```
public DateTime()
    throws ExitFuncException
```

引数

なし

説明

デフォルトの日付時刻を生成します。年月日分秒に現在の日付・時刻を，タイムゾーン文字列に現在のこのホストのデフォルトのタイムゾーン文字列（TimeZone.getDefault().getDisplayName() の値）を指定して，日付時刻を生成する場合と等価です。

例外

例外	内容
ExitFuncException	日付・時刻の設定に失敗した場合に発生します。

（3） メソッドの詳細

（a） getYear

形式

```
public int getYear()
```

引数

なし

説明

年を取得します。

戻り値

年が返されます。

(b) setYear

形式

```
public void setYear(int year)
    throws ExitFuncException
```

引数

引数	内容
year	年を指定します。

説明

指定の年を設定します。

戻り値

なし

例外

例外	内容
ExitFuncException	引数に範囲外の値を指定した場合に発生します。

(c) getMonth

形式

```
public int getMonth()
```

引数

なし

説明

月を取得します。

戻り値

月が返されます。

(d) setMonth

形式

```
public void setMonth(int month)
    throws ExitFuncException
```

引数

引数	内容
month	月を指定します。

説明

指定の月を設定します。

戻り値

なし

例外

例外	内容
ExitFuncException	引数に範囲外の値を指定した場合に発生します。

(e) getDay

形式

```
public int getDay()
```

引数

なし

説明

日付を取得します。

戻り値

日付が返されます。

(f) setDay

形式

```
public void setDay(int day)
    throws ExitFuncException
```

引数

引数	内容
day	日付を指定します。

説明

指定の日付を設定します。

戻り値

なし

例外

例外	内容
ExitFuncException	引数に範囲外の値を指定した場合に発生します。

（g） getHour

形式

```
public int getHour()
```

引数

なし

説明

時を取得します。

戻り値

時が返されます。

（h） setHour

形式

```
public void setHour(int hour)
    throws ExitFuncException
```

引数

引数	内容
hour	時を指定します。

説明

指定の時を設定します。

戻り値

なし

例外

例外	内容
ExitFuncException	引数に範囲外の値を指定した場合に発生します。

(i) getMinute

形式

```
public int getMinute()
```

引数

なし

説明

分を取得します。

戻り値

分が返されます。

(j) setMinute

形式

```
public void setMinute(int minute)
    throws ExitFuncException
```

引数

引数	内容
minute	分を指定します。

説明

指定の分を設定します。

戻り値

なし

例外

例外	内容
ExitFuncException	引数に範囲外の値を指定した場合に発生します。

(k) getSecond

形式

```
public double getSecond()
```

引数

なし

説明

秒を取得します。

戻り値

秒を取得します。

(l) setSecond

形式

```
public void setSecond(double second)
    throws ExitFuncException
```

引数

引数	内容
second	秒を指定します。

説明

指定の秒を設定します。

戻り値

なし

例外

例外	内容
ExitFuncException	引数に範囲外の値を指定した場合に発生します。

(m) getZone

形式

```
public String getZone()
```

引数

なし

説明

タイムゾーン文字列を取得します。

戻り値

タイムゾーン文字列が返されます。

（n） setZone

形式

```
public void setZone(String timeZone)
```

引数

引数	内容
timeZone	タイムゾーン文字列を指定します。

説明

指定のタイムゾーン文字列を設定します。

戻り値

なし

（o） getDate

形式

```
public Date getDate()
```

引数

なし

説明

設定済みの日付・時刻を表す Date オブジェクトを取得します。プラットホームデフォルトのロケールとタイムゾーンに従って，java.util.Date オブジェクトを生成して，その結果を返します。ただし，年に 0 を指定した場合は，生成される java.util.Date オブジェクトは 0001 年になります。

戻り値

設定済みの日付・時刻を表す Date オブジェクトが返されます。

10.3.3 ExitFuncWarning クラス

ここでは，ExitFuncWarning クラスの仕様について説明します。

（1） ExitFuncWarning クラスの概要

継承

```
java.lang.Object
```

```
|
+-- java.lang.Throwable
    |
    +-- java.lang.Exception
        |
        +-- jp.co.Hitachi.soft.interschema2.UserException
            |
            +--
jp.co.Hitachi.soft.interschema2.exitfunc.ExitFuncWarning
```

形式

```
public final class ExitFuncWarning
extends jp.co.Hitachi.soft.interschema2.UserException
```

説明

出口関数で発生したワーニングを定義します。Java 言語の出口関数で発生したワーニングを，このクラスをスローして Interschema に通知します。

出口関数でこのワーニングが発生した場合，Interschema は，出口関数の戻り値として，このクラスに設定された値を仮定して，処理を続行します。

コンストラクタの概要

形式	機能
public ExitFuncWarning(int userErrorCode)	指定されたエラーコード及びデフォルトの戻り値を持つ警告を生成します。
public ExitFuncWarning(int userErrorCode, int result)	指定されたエラーコード及び戻り値を持つ警告を生成します。
public ExitFuncWarning(int userErrorCode, double result)	指定されたエラーコード及び戻り値を持つ警告を生成します。
public ExitFuncWarning(int userErrorCode, String result)	指定されたエラーコード及び戻り値を持つ警告を生成します。
public ExitFuncWarning(int userErrorCode, byte[] result)	指定されたエラーコード及び戻り値を持つ警告を生成します。
public ExitFuncWarning(int userErrorCode, DateTime result)	指定されたエラーコード及び戻り値を持つ警告を生成します。

（2）コンストラクタの詳細

(a) ExitFuncWarning

形式

```
public ExitFuncWarning(int userErrorCode)
```

引数

引数	内容
userErrorCode	エラーコードを指定します。

説明

指定されたエラーコード及びデフォルトの戻り値を持つ警告を生成します。このコンストラクタで生成した警告をスローすると，出口関数の型に対応したデフォルトの値を戻り値に使用します。出口関数の型に対応するデフォルトの戻り値を次の表に示します。

表 10-4　出口関数の型に対応する警告時のデフォルト戻り値

出口関数の型	デフォルト値
int	0
double	0
java.lang.String	" "（半角スペース）
byte[]	要素 0，長さ 1 の配列
jp.co.Hitachi.soft.interschema2.exitfunc.DateTime	現在時刻（引数なしのコンストラクタで生成された DateTime オブジェクト）

（b） ExitFuncWarning

形式

```
public ExitFuncWarning(int userErrorCode, int result)
```

引数

引数	内容
userErrorCode	エラーコードを指定します。
result	戻り値を指定します。

説明

指定されたエラーコードと戻り値を持つ警告を生成します。

出口関数の型が int 型の場合は，このコンストラクタで生成した警告をスローすると，戻り値として result に設定した値が使用されます。

出口関数の型が int 型でない場合は，このコンストラクタで生成した警告をスローすると，出口関数の型に対応したデフォルトの値を戻り値に使用します。デフォルトの値については，「表 10-4　出口関数の型に対応する警告時のデフォルト戻り値」を参照してください。

（c） ExitFuncWarning

形式

```
public ExitFuncWarning(int userErrorCode, double result)
```

引数

引数	内容
userErrorCode	エラーコードを指定します。
result	戻り値を指定します。

説明

指定されたエラーコードと戻り値を持つ警告を生成します。
出口関数の型が double 型の場合は，このコンストラクタで生成した警告をスローすると，戻り値として result に設定した値が使用されます。出口関数の型が double 型でない場合は，このコンストラクタで生成した警告をスローすると，出口関数の型に対応したデフォルトの値を戻り値に使用します。デフォルトの値については，「表 10-4　出口関数の型に対応する警告時のデフォルト戻り値」を参照してください。

（d） ExitFuncWarning

形式

```
public ExitFuncWarning(int userErrorCode, String result)
```

引数

引数	内容
userErrorCode	エラーコードを指定します。
result	戻り値を指定します。

説明

指定されたエラーコードと戻り値を持つ警告を生成します。
出口関数の型が String 型の場合は，このコンストラクタで生成した警告をスローすると，戻り値として result に設定した値が使用されます。
出口関数の型が String 型でない場合は，このコンストラクタで生成した警告をスローすると，出口関数の型に対応したデフォルトの値を戻り値に使用します。デフォルトの値については，「表 10-4　出口関数の型に対応する警告時のデフォルト戻り値」を参照してください。

（e） ExitFuncWarning

形式

```
public ExitFuncWarning(int userErrorCode, byte[] result)
```

引数

引数	内容
userErrorCode	エラーコードを指定します。
result	戻り値を指定します。

説明

指定されたエラーコードと戻り値を持つ警告を生成します。
出口関数の型が byte[] 型の場合は，このコンストラクタで生成した警告をスローすると，戻り値として result に設定した値が使用されます。出口関数の型が byte[] 型でない場合，このコンストラクタで生成した警告をスローすると，出口関数の型に対応したデフォルトの値を戻り値に使用します。デフォルトの値については，「表 10-4　出口関数の型に対応する警告時のデフォルト戻り値」を参照してください。

(f) ExitFuncWarning

形式

```
public ExitFuncWarning(int userErrorCode, DateTime result)
```

引数

引数	内容
userErrorCode	エラーコードを指定します。
result	戻り値を指定します。

説明

指定されたエラーコードと戻り値を持つ警告を生成します。
出口関数の型が DateTime 型の場合は，このコンストラクタで生成した警告をスローすると，戻り値として result に設定した値が使用されます。出口関数の型が DateTime 型でない場合，このコンストラクタで生成した警告をスローすると，出口関数の型に対応したデフォルトの値を戻り値に使用します。デフォルトの値については，「表 10-4　出口関数の型に対応する警告時のデフォルト戻り値」を参照してください。

10.3.4 ExitFuncException クラス

ここでは，ExitFuncException クラスの仕様について説明します。

(1) ExitFuncException クラスの概要

継承

```
java.lang.Object
|
+-- java.lang.Throwable
    |
```

```
    +-- java.lang.Exception
        |
        +-- jp.co.Hitachi.soft.interschema2.UserException
            |
            +--
jp.co.Hitachi.soft.interschema2.exitfunc.ExitFuncException
```

形式

```
public final class ExitFuncException
extends jp.co.Hitachi.soft.interschema2.UserException
```

説明

出口関数で発生したエラー例外を定義します。Java 言語の出口関数で発生したエラーを，この例外をスローして Interschema に通知します。

出口関数でこの例外が発生した場合，Interschema は，異常終了したものとして処理を終了します。この場合，Translator.exec 関数には TranslatorException がスローされます。

コンストラクタの概要

形式	機能
public ExitFuncException(int userErrorCode)	指定されたエラーコードを持つ出口関数例外を生成します。

（2） コンストラクタの詳細

（a） ExitFuncException

形式

```
public ExitFuncException(int userErrorCode)
```

引数

引数	内容
userErrorCode	エラーコードを指定します。

説明

指定されたエラーコードを持つ出口関数例外を生成します。

10.4 データ変換処理 API（Java 言語）の使用例

この節では，データ変換 API（Java 言語）の使用例を示します。

データ変換 API（Java 言語）の使用例には，次の三つの例があります。

- サンプルコード（サーブレット版）
- サンプルコード（TranslateData 派生クラス）
- サンプルコード（DateTime クラス使用例）

(1) サンプルコード（サーブレット版）

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.LinkedList;
import java.util.List;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import jp.co.Hitachi.soft.interschema2.*;

/**
 * タイトル:  SampleServlet
 * 説明:    uCosminexus Interschema V2 Java API Servlet Sample.
 * 著作権:  All Rights Reserved. Copyright (C) 2005, Hitachi,Ltd.
 * @version  02-01
 */
public class SampleServlet extends HttpServlet {

    public static final String DOCTYPE =
        "<!DOCTYPE HTML PUBLIC ¥"-//W3C//DTD HTML 4.0 " +
        "Transitional//EN¥"";

    public static String headWithTitle(String title) {
        return DOCTYPE + "¥n" +
            "<HTML>¥n" +
            "<HEAD><TITLE>" + title + "</TITLE></HEAD>¥n";
    }

    /**
     *
     * @param request request クライアントがサーブレットに行う要求を含む
HttpServletRequest オブジェクト
     * @param response response サーブレットがクライアントに送信する応答を
含む HttpServletResponse オブジェクト
    * @see javax.servlet.http.HttpServlet#doGet(HttpServletRequest,
HttpServletResponse)
     */
    protected void doGet(HttpServletRequest request,
HttpServletResponse response)
                throws ServletException, IOException {
```

```
        /*トランスレータ変数の定義*/
        Translator trans = null;
        /*MDL情報変数の定義*/
        MDLInfo mdlInfo = null;

        ServletContext sc = getServletContext();

        sc.log("変換を開始します。");
        try{
            /* パラメータの取得 */
            String inputFormat = request.getParameter("inFormat");
            String inputData = request.getParameter("inputData");
            String outFormat = request.getParameter("outFormat");
            String mdlFileName = request.getParameter("mdlFile");

            /*変換ライブラリの構築*/
            trans = new Translator("Sample.log");

            /*MDLファイルのロード*/
            mdlInfo = new MDLInfo(mdlFileName);

            /*入出力フォーマット情報の作成*/
            List input = new LinkedList();
            List output = new LinkedList();

            input.add(new StringData(inputFormat, inputData));
            output.add(new StringData(outFormat, null));

            /*変換処理の実行*/
            trans.exec(mdlInfo, input, output);

            String[] outData = new String[output.size()];
            for (int i = 0; i < output.size(); i++) {
             TranslateData tranData = (TranslateData)output.get(i);
                outData[i] = (String)tranData.getDataObject();
            }

            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
            out.print(headWithTitle("Interscheama 02-01 Servlet
Sample"));
            for (int i = 0; i < outData.length; i++) {
                out.print(outData[i]);
            }

            /*変換終了ログの出力*/
            sc.log("変換が正常終了しました。");
        } catch (UserException ue) {
            sc = getServletContext();
            sc.log("UserExceptionが発生しました。", ue);
        } catch(TranslatorException te) {
            sc = getServletContext();
            sc.log("TranslatorExceptionが発生しました。[0x"
                     + Integer.toHexString(te.getErrorCode()) + "]"
                    + getErrorString(te), te);
        }
    }

    /**
     * サーブレットが POST 要求を処理する際に，service メソッド経由でサーバ
によって呼び出されます。
     * POST 要求も GET 要求と同じ処理を行う。
```

```
     * @param request クライアントがサーブレットに行う要求を含む
HttpServletRequest オブジェクト
     * @param response サーブレットがクライアントに送信する応答を含む
HttpServletResponse オブジェクト
     * @see
javax.servlet.http.HttpServlet#doPost(HttpServletRequest,
HttpServletResponse)
     */
    protected void doPost(HttpServletRequest request,
HttpServletResponse response)
                throws ServletException, IOException {
        this.doGet(request, response);
    }

    /**
     * 拡張エラー情報を取得して，文字列に変換します。
     * @param te 拡張エラー情報文字列を取得するためのTranslatorException
     * @return 拡張エラー情報文字列
     */
    static String getErrorString(TranslatorException te) {
        ExtraErrorData d[] = te.getExtraErrData();

        if (d == null) {
            return new String("");
        }
        StringBuffer msg = new StringBuffer();
        for(int i = 0; i < d.length; i++) {
            msg.append("エラー情報[" + i + "]:");
            String s = Integer.toString(d[i].getMessageNo());
            msg.append("メッセージ番号[" + s + "],");
            s = Integer.toHexString(d[i].getErrorLV());
            msg.append("エラーレベル[0x" + s + "],");
            s = Integer.toString(d[i].getNumericData());
            msg.append("数値情報[" + s + "],");
         msg.append("文字列情報[" + d[i].getInformation() + "]");
        }
        return msg.toString();
    }
}
```

（2）サンプルコード（TranslateData 派生クラス）

```
import jp.co.Hitachi.soft.interschema2.TranslateData;
import jp.co.Hitachi.soft.interschema2.UserException;

/**
 * タイトル:  ByteArrayData
 * 説明:    uCosminexus Interschema V2 Java API Sample.
 * 著作権:  All Rights Reserved. Copyright (C) 2005, Hitachi,Ltd.
 * @version  02-01
 */
public class ByteArrayData extends TranslateData {

    /**
     * 指定のフォーマット名とバイト配列からバイト配列形式の変換対象データを生成
する。
     * @param formatName 変換対象フォーマット名
```

```
     * @param formatData バイト配列形式のデータオブジェクト
     */
    public ByteArrayData(String formatName, byte[] dataObj) {
        super.setFormatName(formatName);
        super.setDataObject(dataObj);
    }

    /**
     * 指定のデータオブジェクトをバイト配列に変換する。
     * @param dataObj データオブジェクト
     * @exception UserException 指定のデータオブジェクトがバイト配列でない
場合
         * @see
jp.co.Hitachi.soft.interschema2.TranslateData#toByteArray(Object)
     */
    protected byte[] toByteArray(Object dataObj)
                throws UserException {
        if (!(dataObj instanceof byte[])) {
            throw new UserException();
        }
        return (byte[])dataObj;
    }

    /**
     * 指定のバイト配列をデータオブジェクトに変換する。
     * 指定のバイト配列をそのまま返します。
     * @param byteData バイト配列形式のデータオブジェクト
     * @exception TranslatorException 発生しない。
        * @see
jp.co.Hitachi.soft.interschema2.TranslateData#toDataObject(byte[])
     */
    protected Object toDataObject(byte[] byteData) {
                return byteData;
    }
}
```

（3）サンプルコード（DateTime クラス使用例）

```
package sample;

import java.util.Date;
import java.text.DateFormat;
import java.text.ParseException;
import jp.co.Hitachi.soft.interschema2.exitfunc.DateTime;
import jp.co.Hitachi.soft.interschema2.exitfunc.ExitFuncException;

/**
 *Javaによる出口関数を定義します。
 */
public class UserFunction {
    /**
     * 日付時刻を表す文字列をJavaメソッド，DateFormat.parse
     * を利用して解析し，Interschemaの日付時刻形式として設定
     * します。
     * この関数のettrans.iniの定義は以下のようになります。
     * ET_DTM FUNC002 (ET_STRING)
     * 第一引数のObjectは明示的には記述しません。
```

```
     * 関数名FUNC002は任意に付けた名前であり，Java
     * メソッド名と一致させる必要はありません。
     */
    public static DateTime func002(Object o, String strDateTime)
                throws ExitFuncException {
        DateTime result = null;
        try {
            /* デフォルトの日付フォーマッタを */
            /* 取得します。 */
            DateFormat dtFmt =
                 DateFormat.getDateTimeInstance();
            /* 入力日付文字列を解析します。 */
            Date date = dtFmt.parse(strDateTime);
            result = new DateTime(date);
        } catch (ParseException e) {
            /* 変換エラー発生                */
            throw new ExitFuncException(1);
        }       return result;
    }
}
```

11 ユーザ組み込み関数

この章では，ユーザ組み込み関数及びユーザ組み込み関数が呼び出す出口関数について説明します。

11.1　ユーザ組み込み関数の概要

Interschema では，変換情報を定義するときに，標準で提供している関数以外に，ユーザが独自に定義した関数を使用できます。このような関数を「ユーザ組み込み関数」と呼びます。

この節では，ユーザ組み込み関数について説明します。

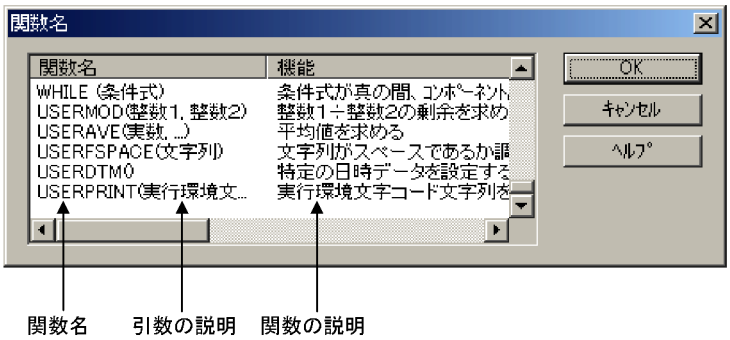
ユーザ組み込み関数の特長は，次のとおりです。

- ユーザ組み込み関数は，999 個まで定義できます。
- 任意の名前のユーザ組み込み関数を定義できます。
- 可変個引数の関数が定義できます。
- 出口となる C 言語又は Java 言語の関数は，任意のライブラリに定義できます。

FDL エディタ及び MDL エディタの式中に関数を使用するとき，使用する関数を［関数名］ダイアログから選択できます。

ユーザ組み込み関数を定義すると，［関数名］ダイアログにユーザ組み込み関数が表示されます。［関数名］ダイアログには，組み込んだ関数の関数名，引数の説明，及び関数の説明が表示できます。［関数名］ダイアログの例を次に示します。

図 11-1　［関数名］ダイアログの例



また，組み込んだ関数は，FDL エディタ及び MDL エディタで式を指定するテキスト内に直接記述できます。

ユーザ組み込み関数は，C 言語又は Java 言語で作成したユーザプログラムへの出口として機能します。ユーザ組み込み関数の機能は，出口関数（ユーザが作成する C 言語又は Java 言語の関数のことで，トランスレータから制御を渡す際の出口になるもの）中に記述します。

11.2 ユーザ組み込み関数の設定手順

ユーザ組み込み関数の設定手順を次に示します。

C 言語の出口関数を使用する場合

1. **システム情報ファイル「ettrans.ini」の定義**
   システム情報ファイル「ettrans.ini」に，ユーザ組み込み関数の情報や出口関数の情報を定義します。
2. **出口関数の定義**
   ユーザ組み込み関数を記述した出口関数を定義します。
3. **ダイナミックリンクライブラリ（DLL）（Windows の場合）又は共用ライブラリ（ワークステーションの OS の場合）の作成**
   DLL（Windows の場合）又は共用ライブラリ（ワークステーションの OS の場合）を作成します。

Java 言語の出口関数を使用する場合

1. **システム情報ファイル「ettrans.ini」の定義**
   システム情報ファイル「ettrans.ini」に，ユーザ組み込み関数の情報や出口関数の情報を定義します。
2. **出口関数の定義**
   ユーザ組み込み関数を記述した出口関数を定義します。
3. **Java クラスファイルの設定**
   出口関数を定義した Java クラスファイルを，Java 実行環境のクラスパス内に配置します。

11.3 システム情報ファイル「ettrans.ini」の定義

ユーザ組み込み関数は，システム情報ファイル「ettrans.ini」に定義します。

ワークステーションの OS の場合，システム情報ファイルは，システムがサポートする実行環境のロケールに対応したコードで記述してください。

システム情報ファイル「ettrans.ini」で定義が必要なセクションを次に示します。

表 11-1 システム情報ファイル「ettrans.ini」で定義が必要なセクション

セクション	トランスレータ			FDL エディタ及び MDL エディタ
	C 言語の出口関数（Windows の場合）	C 言語の出口関数（ワークステーションの OS の場合）	Java 言語の出口関数	
Userfunc_MaxId	○	○	○	○
Userfunc_Mapfunc	○	○	○	○
Userfunc_MapfuncGuide	－	－	－	○
Userfunc_ExitfuncName	○	○	－	－
Userfunc_DllPath	○	－	－	－
Userfunc_SlPath	－	○	－	－
Userfunc_JavaMethodName	－	－	○	△
Userfunc_Option	－	－	－	－

（凡例）

○：定義が必要なセクションです。

△：Java 言語の出口関数を使用する場合に必要なセクションです。

－：該当しません（任意です）。

注

- [Userfunc_JavaMethodName] セクションの下に有効なメソッド名が定義された関数 ID を持つユーザ組み込み関数は，Java 言語の出口関数として扱われます。
- それ以外のユーザ組み込み関数は，C 言語の出口関数として扱われます。
- ただし，[Userfunc_ExitfuncName] セクション，又は [Userfunc_JavaMethodName] セクションに同じ関数 ID を持つユーザ組み込み関数定義することはできません。

システム情報ファイル「ettrans.ini」の記述例を次に示します。

● システム情報ファイル「ettrans.ini」の記述例（C 言語の関数を定義する場合）

```
; ettrans.ini

[Userfunc_MaxId]
MAX=005

[Userfunc_Mapfunc]
FUNC001=ET_INT USERMOD(ET_INT,ET_INT)
FUNC002=ET_REAL USERAVE(ET_REAL,...)
FUNC003=ET_STRING USERFSPACE(ET_STRING)
FUNC004=ET_DTM USERDTM()
FUNC005=ET_STRING USERPRINT(ET_ENVSTRING)

[Userfunc_MapfuncGuide]
FUNC001=(整数１, 整数２);整数１÷整数２の剰余を求める
FUNC002=(実数, ...);平均値を求める
FUNC003=(文字列);文字列がスペースであるか調べる
FUNC004=();特定の日時データを設定する
FUNC005=(実行環境文字コード文字列);実行環境文字コード文字列をファイルに出
力する

[Userfunc_ExitfuncName]
FUNC001=usermod
FUNC002=useraverage
FUNC003=userisspace
FUNC004=userdtm
FUNC005=userPrintEnvstring

[Userfunc_DllPath]
FUNC001=c:¥interschema2¥bin¥USERFUNC.dll
FUNC002=c:¥interschema2¥bin¥USERFUNC.dll
FUNC003=c:¥interschema2¥bin¥USERFUNC.dll
FUNC004=c:¥interschema2¥bin¥USERFUNC.dll
FUNC005=c:¥interschema2¥bin¥USERFUNC.dll

[Userfunc_SlPath]
FUNC001=/opt/hitachi/interschema/lib/libuser.sl
FUNC002=/opt/hitachi/interschema/lib/libuser.sl
FUNC003=/opt/hitachi/interschema/lib/libuser.sl
FUNC004=/opt/hitachi/interschema/lib/libuser.sl
FUNC005=/opt/hitachi/interschema/lib/libuser.sl

[Userfunc_Option]
FUNC005=USE02IF
```

● システム情報ファイル「ettrans.ini」の記述例（Java 言語の関数を定義する場合）

```
; ettrans.ini

[Log]
Size=1000

[Userfunc_MaxId]
MAX=8
```

```
[Userfunc_Mapfunc]
FUNC001=ET_INT F001(ET_INT,ET_INT)
FUNC002=ET_INT F002(ET_DTM)
FUNC003=ET_INT F003(ET_STREAM)
FUNC004=ET_INT F004(ET_ENVSTRING)
FUNC005=ET_STRING F005(ET_STREAM)
FUNC006=ET_STREAM F006(ET_ENVSTRING)
FUNC007=ET_STRING F007(ET_DTM)
FUNC008= ET_INT F008(ET_REAL,ET_REAL)

[Userfunc_MapfuncGuide]
FUNC001=(引数1,引数2);Java Method 001
FUNC002=(日付時刻文字列);Java Method 002
FUNC003=;Java Method 003
FUNC004=;Java Method 004
FUNC005=;Java Method 005
FUNC006=;Java Method 006
FUNC007=;Java Method 007
FUNC008=;Java Method 008

[Userfunc_JavaMethodName]
FUNC001= usrpackagename.UFunc.func001
FUNC002= usrpackagename.UFunc.func002
FUNC003= usrpackagename.UFunc.func003
FUNC004= usrpackagename.UFunc.func004
FUNC005= usrpackagename.UFunc.func005
FUNC006= usrpackagename.UFunc.func006
FUNC007= usrpackagename.UFunc.func007
FUNC008= usrpackagename.UFunc.func008
```

システム情報ファイル「ettrans.ini」の各セクションについて次に説明します。

11.3.1 [Userfunc_MaxId]セクション

組み込み関数の数を MAX キーで定義します。

(1) 書式

MAX= 最大 ID

(2) 説明

最大 ID

組み込み関数の最大 ID を定義します（エントリ数ではないので注意してください）。
001 から 999 まで指定できます。先頭の 0 は省略できます。
C 言語の関数及び Java 言語の関数を混在させている場合は，C 言語の関数及び Java 言語の関数を併せた中での最大 ID を指定します。

11.3.2 [Userfunc_Mapfunc]セクション

組み込み関数のインタフェース（戻り値，引数）を FUNC*nnn* キーで定義します。組み

込み関数の定義数分だけキー定義の行を記述します。

(1) 書式

FUNC*nnn*= 戻り値の型△名前（引数 1 の型，引数 2 の型，...）

注：△は，半角スペースを表しています。

(2) 説明

FUNC*nnn*

nnn には，組み込み関数の ID を指定します。組み込み関数の ID は，001 から 999 まで指定できます。組み込み関数の ID は必ず 3 桁で指定してください。先頭の 0 は省略できません。ここで指定した ID は，ほかのセクションで指定する ID と一致させる必要があります。

行を ID 順にソートする必要はありません。また，ID は連続していなくてもかまいません。ただし，必ず最大 ID が [Userfunc_MaxId] セクションの MAX キーに定義されている必要があります。

戻り値の型

組み込み関数の戻り値の型を指定します。戻り値の型は，「表 11-2　ユーザ組み込み関数で使用する型」に示す型の中から指定する必要があります。ただし，ET_PTR 型，ET_ENVSTRING 型は指定できません。

名前

組み込み関数の名前を指定します。名前には，半角英数字だけを使用できます。ただし，名前の先頭文字は英字にする必要があります。長さは 64 バイト以内で指定します。

ここで指定した組み込み関数の名前が，FDL エディタ及び MDL エディタの［関数名］ダイアログの関数リストの関数名欄に表示されます。

名前の指定では，Interschema が標準で提供している関数名やほかのユーザ組み込み関数名と同じ名前は指定できません。

関数名の先頭文字は，次のどれかの文字にしてください。

U，u，W，w，X，x，Y，y，Z，z

Interschema が標準で提供している関数については，「6.7.3　関数」を参照してください。

引数の型

組み込み関数の引数の型を指定します。引数の型は，「表 11-2　ユーザ組み込み関数で使用する型」に示す型の中から指定する必要があります。ただし，ET_PTR 型は指定できません。また，Java 言語の出口関数の場合，ET_STRING 型は指定できません。

引数が複数個ある場合は，引数の数だけ半角コンマで区切って記述します。引数がない場合は，括弧だけを記述します。引数が可変個の場合は，繰り返し部分の先頭の型の後に「...」を記述します。ただし，可変個引数の繰り返し部分は引数の最後

に記述する必要があります。また，繰り返し部分はすべて同じ型にする必要があります。可変個引数の定義例を，次に示します。

```
FUNC001=ET_INT USRFNC(ET_STRING,ET_INT,...※)
```

注※　...は，ET_INT型の引数の繰り返しを表します。

ユーザ組み込み関数で使用する型

ユーザ組み込み関数で使用する型を次に示します。戻り値の型，及び引数の型は，次に示す型の中から指定してください。

表 11-2　ユーザ組み込み関数で使用する型

型	意味	備考
ET_INT	整数	C 言語の long 型（32bit）として表現できる範囲の整数です。
ET_REAL	実数	C 言語の double 型（32bit）として表現できる範囲の実数です。
ET_STRING	文字列	－
ET_STREAM	バイト列	－
ET_DTM	日付時刻	日付，時刻，日付時刻型に対応します。
ET_PTR	ポインタ型	出口関数引数だけで使用され，ユーザ組み込み関数の引数，及び戻り値では使用できません。
ET_ENVSTRING	実行環境文字コード文字列型	実行環境の文字コードに変換された文字列です。ユーザ組み込み関数の引数だけで使用できます。

（凡例）

－：該当しません。

(3) 注意

- このセクションを省略した場合，ユーザ組み込み関数は使用できません。
- FUNC*nnn* キーを省略した場合や記述に不備があった場合，そのキーで定義した組み込み関数は使用できません。
- 「戻り値の型」と「名前」の間には，半角スペースを一文字入れる必要があります。その他の場所にはスペースを入れないでください。
- 「=」の右辺の文字列は，255 バイト以内で記述する必要があります。
- ユーザ組み込み関数の引数として指定できる要素は，標準で提供している関数と同じです。実引数の型と組み込み関数の引数の型が異なる場合の変換処理も標準組み込み関数と同じ規則に従います。ただし，整数型（ET_INT）の引数に小数が入力される場合は，小数部を切り捨てた整数に変換します。
- 整数型（ET_INT）引数に C 言語の long 型で収まらない範囲の整数が入力された場合や，実数型（ET_REAL）引数に double 型で収まらない範囲の実数が入力された場合は，変換時にエラーとなります。日付時刻型 (ET_DTM) についても同様です。
- C 言語の出口関数に対応する組み込み関数で，実行環境文字コード文字列型

（ET_ENVSTRING）を使用する場合，[Userfunc_Option] セクションでオプション USE02IF の指定が必要です。Java 言語の出口関数に対応する組み込み関数では，このオプションの指定は不要です。

11.3.3 ［Userfunc_MapfuncGuide］セクション

関数ダイアログの関数リストに表示する組み込み関数の説明を FUNC*nnn* キーで定義します。この定義は，FDL エディタ又は MDL エディタだけで使用します。

ここで指定した引数の説明と関数の説明が，［関数名］ダイアログの関数名欄と機能欄に表示されます。

（1） 書式

FUNC*nnn*= 引数の説明 ; 関数の説明

（2） 説明

FUNC*nnn*

nnn には，[Userfunc_Mapfunc] セクションで定義した組み込み関数の ID と同じ ID を指定します。

引数の説明

組み込み関数の引数の説明を記述します。ただし，半角セミコロン「;」は使用できません。

関数の説明

関数の意味や仕様を記述します。ただし，半角セミコロン「;」は使用できません。

（3） 注意

- このセクションを省略した場合は，関数リスト中のユーザ組み込み関数のすべての説明欄が空白になります。
- FUNC*nnn* キーを省略した場合は，その組み込み関数の説明欄だけが空白になります。
- 引数の説明，又は関数の説明のどちらかの指定を省略した場合は，対応する説明の欄だけが空白になります。ただし，この場合，引数の説明と関数の説明の区切りのセミコロン「;」は指定する必要があります。
- 「=」の右辺の文字列は，255 バイト以内で記述する必要があります。

11.3.4 ［Userfunc_ExitfuncName］セクション

出口関数の名前を FUNCnnn キーで定義します。

（1） 書式

FUNC*nnn*= 出口関数名

(2) 説明

FUNC*nnn*

nnn には，[Userfunc_Mapfunc] セクションで定義した組み込み関数の ID と同じ ID を指定します。

出口関数名

出口関数名を指定します。出口関数名は，C 言語の関数名として使用できる範囲で任意の名前を定義してください。ただし，定義済みのほかの出口関数と同じ名前は指定できません。

(3) 注意

- このセクションの指定は，C 言語の出口関数を変換実行するときに必要なものです。
- 「=」の右側の文字列は，255 バイト以内で記述する必要があります。
- 同じ ID の組み込み関数に対して，このセクションと [Userfunc_JavaMethodName] セクションを同時に指定することはできません。

11.3.5 [Userfunc_DllPath] セクション

出口関数を含む DLL のフルパスを FUNC*nnn* キーで定義します。この定義は，Windows で変換する場合に必要です。

(1) 書式

FUNC*nnn*=DLL パス

(2) 説明

FUNC*nnn*

nnn には，[Userfunc_Mapfunc] セクションで定義した組み込み関数の ID と同じ ID を指定します。

DLL パス

[Userfunc_ExitfuncName] セクションで定義した出口関数を含む DLL のフルパスを記述します。

(3) 注意

- このセクションの指定は，C 言語の出口関数を Windows で変換実行する時に必要なものです。
- 同じ DLL に複数の出口関数を定義してもかまいません。
- 「=」の右辺の文字列は，255 バイト以内で記述する必要があります。

11.3.6 [Userfunc_SlPath]セクション

出口関数を含む共用ライブラリのフルパスを FUNC*nnn* キーで定義します。この定義は，ワークステーションの OS で変換する場合に必要です。

(1) 書式

FUNC*nnn*= 共用ライブラリのパス

(2) 説明

FUNC*nnn*

nnn には，[Userfunc_Mapfunc] セクションで定義した組み込み関数の ID と同じ ID を指定します。

共用ライブラリのパス

[Userfunc_ExitfuncName] セクションで定義した出口関数を含む共用ライブラリのフルパスを記述します。

(3) 注意

- このセクションの指定は，C 言語の出口関数をワークステーションの OS で変換実行するときに必要なものです。
- 同じ共用ライブラリに複数の出口関数を定義してもかまいません。
- 「=」の右辺の文字列は，255 バイト以内で記述する必要があります。

11.3.7 [Userfunc_JavaMethodName]セクション

[Userfunc_Mapfunc] セクションで定義したユーザ組み込み関数から呼び出す Java 言語の出口関数のメソッド名を指定します。

(1) 書式

FUNC*nnn*=[パッケージ名 .] クラス名 . メソッド名

(2) 説明

FUNC*nnn*

nnn には，[Userfunc_Mapfunc] セクションで定義した組み込み関数の ID と同じ ID を指定します。

パッケージ名

Java 言語の出口関数が含まれるパッケージ名を記述します。ただし，パッケージ化していない場合は省略できます。

クラス名

Java 言語の出口関数が含まれるクラス名を記述します。

メソッド名

Java 言語の出口関数のメソッド名を記述します。

(3) 注意

- このセクションの指定は，Java 言語の出口関数に対応する組み込み関数の式を FDL エディタ又は MDL エディタで記述するとき，及びその変換実行時に必要なものです。
- ここで指定したクラスは，変換実行時にはデータ変換処理 API（Java 言語）を呼び出す Java 環境のクラスパスに含まれていなければなりません。FDL エディタ又は MDL エディタ使用時はその必要はありません。
- 「=」の右辺の文字列は，255 バイト以内で記述する必要があります。
- 同じ ID の組み込み関数に対して，このセクションと [Userfunc_ExitfuncName] セクションを同時に指定することはできません。

11.3.8 [Userfunc_Option] セクション

出口関数で使用するオプションを指定します。

(1) 書式

FUNC*nnn*= オプション名

(2) 説明

FUNC*nnn*

nnn には，[Userfunc_Mapfunc] セクションで定義した組み込み関数の ID と同じ ID を指定します。

オプション名

出口関数に渡すオプション名を記述します。

USE02IF

出口関数の入力引数に変換実行関数（ETtrans2Exec 関数）で指定されたポインタ（ET_PTR 型）が渡されます。なお，Interschema バージョン 1 の互換 API が使用されている場合は，ポインタとして null が渡されます。
また，実行環境文字コード文字列型（ET_ENVSTRING）を使用できます。実行環境文字コード文字列型（ET_ENVSTRING）については，「11.3.2 [Userfunc_Mapfunc] セクション」の「表 11-2 ユーザ組み込み関数で使用する型」を参照してください。

(3) 注意

- オプション USE02IF を指定していない出口関数には，ポインタ（ET_PTR 型）を渡しません。また，実行環境文字コード文字列型（ET_ENVSTRING）を使用すると，エラーとなります。
- C 言語で新しく出口関数を作成する場合，USE02IF オプションを指定してください。

11.4 C 言語の出口関数の作成

この節では，C 言語出口関数の作成方法について説明します。

11.4.1 C 言語の出口関数の定義

ユーザ組み込み関数の機能を，C 言語の出口関数中に記述します。

（1） C 言語の出口関数の仕様

C 言語の出口関数の仕様を次に示します。

(a) 形式

```
#include  "ETexit.h"

long  userfunc (long lArgNum, EXITARG *ppInArg[],
                EXITARG *pOutArg )
```

注：出口関数名（userfunc部分）には任意の名前を指定します。

(b) 引数

（入力）

lArgNum

ppInArg 配列の要素の数が指定されます。

ppInArg

ユーザ組み込み関数の引数についての情報を格納する ETEXITARG 構造体のポインタ配列が指定されます。ETEXITARG 構造体については，「(c)　説明」を参照してください。

（入出力）

pOutArg

ユーザ組み込み関数の戻り値についての情報を示す，ETEXITARG 構造体のポインタが渡されます。

(c) 説明

ユーザ組み込み関数を使用した場合，トランスレータは，システム情報ファイル「ettrans.ini」の定義に従って対応する出口関数を呼び出します。

lArgNum が 0 の場合は，ppInArg に NULL が設定されます。

ETEXITARG 構造体

ETEXITARG 型は，入力となる引数の情報，及び出力となる戻り値の情報を格納する構造体であり，ヘッダファイル「ETexit.h」内で次のように定義されています。

```
typedef struct {
    long lType;
    ETEXITDATA data;
} ETEXITARG;
```

lType にはデータの型を表す値が入ります。「11.3.2 ［Userfunc_Mapfunc］セクション」の「表 11-2 ユーザ組み込み関数で使用する型」に示した型名をそのままマクロ定数として使用します。これらの値はヘッダファイル「ETexit.h」内で定義されています。
ETEXITDATA 構造体は引数の値を格納するための共用体であり，ヘッダファイル「ETexit.h」内で次のように定義されています。

```
typedef union {
    long lNumber;
    double dNumber;
    void *pPtr;
    ETEXITDATASTRING edstring;
    ETEXITDATASTREAM edstream;
    ETEXITDATADTM eddtm;
    ETEXITDATAENVSTRING edenvstring;
} ETEXITDATA;
```

lNumber は，引数又は戻り値が整数型（ET_INT）の場合に使用します。dNumber は実数型（ET_REAL）の場合に使用します。pPtr はポインタ型（ET_PTR）の場合に使用します。
引数，及び戻り値のそれぞれが，ETEXITDATASTRING 構造体は文字列型（ET_STRING），ETEXITDATASTREAM 構造体はバイト列型（ET_STREAM），ETEXITDATADTM 構造体は日付時刻型（ET_DTM），ETEXITDATAENVSTRING 構造体は実行環境文字コード文字列型（ET_ENVSTRING）の場合に使用します。ヘッダファイル ETexit.h 内で次のように定義されています。

```
typedef struct {
    long lCode;
    long lLength;
    char *pszString;
} ETEXITDATASTRING;

typedef struct {
    long lLength;
    char *psStream;
} ETEXITDATASTREAM;

typedef struct {
    long lYear;         /* year */
    long lMonth;        /* month */
    long lDay;          /* day */
    long lHour;         /* hour */
    long lMinute;       /* minute */
```

```
    double dSecond;     /* second */
    char *pszZone;      /* zone string */
} ETEXITDATADTM;

typedef struct {
    long lErrCode;
    long lCode;
    char *pszString;
} ETEXITDATAENVSTRING;
```

lCode には文字コードを表す定数が入ります。文字コードは，プレフィクス「ETEXIT_CODE_」を付けたマクロで表されます（例：ETEXIT_CODE_JIS8）。これらのマクロは ETexit.h 内に定義されています。

lLength にはデータのサイズ（バイト数）が入ります。データが文字列の場合，終端 NULL はサイズとして数えません。ETEXITDATASTRING 構造体の pszString，及び ETEXITDATASTREAM 構造体の psStream にはそれぞれ文字列，バイト列を格納したメモリエリアへのポインタが入ります。文字列の場合に，終端 NULL は付きません。

ETEXITDATADTM 構造体の lYear ～ dSecond には，対応する日付時刻型のパート値が入ります。日付型や時刻型など，該当するパート値がない場合は，0 を設定します。pszZone には，日付時刻型のゾーン部文字列が終端 NULL 付きで入ります。ゾーン部を持たない場合は NULL です。ゾーン部文字列の文字コードは，Interschema がサポートする実行環境のロケールに従います。Windows の場合は，シフト JIS です。

ETEXITDATAENVSTRING 構造体の pszString には NULL 終端文字列，lErrCode には引数の文字列の実行環境のロケールに従った文字コードへの変換の成否を示すコードが入ります。lErrCode 及び lCode の値と，pszString に格納される文字列の関係を次の表に示します。

表 11-3　ETEXITDATAENVSTRING 構造体のメンバの内容

項番	lErrCode	文字コード変換結果	lCode	pszString
1	0	変換に成功した※1	実行環境のロケールに従った文字コード	実行環境のロケールに従った文字コードの NULL 終端文字列です。
2	0x01000001	変換時に不正文字コードを検出した※2	実行環境のロケールに従った文字コード	実行環境のロケールに従った文字コードの NULL 終端文字列で，検出した不正文字コードをスペース文字へ置換したものです。
3	-1	変換に失敗した	ユーザ組み込み関数に渡された文字列の文字コード	ユーザ組み込み関数に渡された文字列に終端 NULL を付加したものです。

注※ 1

ユーザ組み込み関数に渡された文字列が，最初から実行環境のロケールに従った文字コードの場合も含みます。

注※ 2

フォーマットの指定，又は実行時オプションで不正文字コードのスペース置換を禁止している場合，変換時に不正文字コードを検出したときは，項番 3 の結果が渡されます。

システム情報ファイルの [Userfunc_Option] セクションで USE02IF オプションが指定されている場合，引数 ppInArg の末尾に ET_PTR 型の情報を追加して渡します。ここには ETtrans2Exec 関数で指定されたポインタを格納します。指定がない又はバージョン 1 の互換 API を使用している場合は，NULL が設定されます。また，lArgNum の値はシステム情報ファイルで定義した入力パラメタ数＋ 1 となります。

USE02IF オプションが指定されていない場合は，ET_PTR 型のデータは追加設定しません。

pOutArg のデータは初期化された状態で出口関数に渡されます。ETEXITARG 構造体の lType には，組み込み関数の戻り値型に対応した値があらかじめ設定されます。また，ETEXITDATA 構造体の lNumber，dNumber，ETEXTITDATADTM の lYear 〜 dSecond は，それぞれ 0 で初期化されます。

戻り値の型が文字列型の場合の pszString，バイト列型の場合の psStream，日付時刻型の場合の pszZone には，2,000 バイト分確保したメモリのアドレスが設定されます。2,000 バイトの領域は 0 で初期化されます。文字列，バイト列，日付時刻型のゾーン文字列を出力する場合は，出口関数内で 2,000 バイトに収まるようにデータを設定する必要があります。ゾーン文字列には終端 NULL を付けてください。

異常終了時の戻り値（0 以外の値）は出口関数内で任意に設定してかまいません。負の値を返した場合，トランスレータは変換処理を中止します。正の値を返した場合，トランスレータは変換処理を続行します。続行する場合，pOutArg として出口関数からトランスレータに渡す値はリターン時のものになるので注意してください。エラー発生時の戻り値は，トランスレータのログとして出力されます。

(d) 戻り値

0

正常に終了しました。

＞ 0

異常で終了しました。ただし，トランスレータの処理は続行します。

＜ 0

異常で終了しました。トランスレータの処理を終了します。

(2) C 言語の出口関数の実装例

C 言語の出口関数の実装例を，整数型の場合，実数型で可変個引数の場合，文字列型の場合，日付時刻型の場合，及び USE02IF オプションを使用した場合に分けて次に示します。

● C 言語の出口関数の実装例（整数型）

```
/*
 *  usermod.c -  整数型ユーザ出口関数の実装例
 */
#include  <stdlib.h>
#include  "ETexit.h"

long  usermod(long lArgNum, ETEXITARG *ppInArg[],
              ETEXITARG *pOutArg)
{
     /* 余剰を求める */
     long lArg1, lArg2;

     if (lArgNum != 2 || ppInArg == NULL) return -1;
    if (ppInArg[0] == NULL || ppInArg[0]->lType != ET_INT)  return
-2;
    if (ppInArg[1] == NULL || ppInArg[1]->lType != ET_INT)  return
-2;

     lArg1 = ppInArg[0]->data.lNumber;
     lArg2 = ppInArg[1]->data.lNumber;
     if (lArg2 == 0) return -3;
     pOutArg->data.lNumber = lArg1 % lArg2;

     return 0;
}
```

● C 言語の出口関数の実装例（実数型で可変個引数型）

```
/*
 *  useraverage.c : 実数型可変個引数ユーザ関数の実装例
 */

#include  <stdlib.h>
#include  "ETexit.h"

long  useraverage(long lArgNum, ETEXITARG *ppInArg[],
                  ETEXITARG *pOutArg)
{
    /* 平均値を求める */
    double dTotal = 0.0;
    int i;

    if (lArgNum < 1 || ppInArg == NULL) return -1;
    for ( i = 0; i < lArgNum; i++){
        if (ppInArg[i] == NULL || ppInArg[i]->lType != ET_REAL)
                return -2;
        dTotal += ppInArg[i]->data.dNumber;
    }
    pOutArg->data.dNumber = dTotal/(double)lArgNum;

    return 0;
}
```

● C 言語の出口関数の実装例（文字列型）

```
/*
 *  userisspace.c : 文字列型ユーザ関数の実装例
 */

#include  <stdlib.h>
#include  <string.h>
#include  "ETexit.h"

long  userisspace(long lArgNum, ETEXITARG *ppInArg[],
                  ETEXITARG *pOutArg)
{
    /* 文字列中のスペースをチェックする */
    ETEXITDATASTRING *peds1, *peds2;
    long i;

    if (lArgNum != 1 || ppInArg == NULL)  return -1;
    if (ppInArg[0] == NULL || ppInArg[0]->lType != ET_STRING)
            return -2;
    peds1 = &ppInArg[0]->data.edstring;
    if (peds1->lLength >= 2000) return -3;
    if (peds1->lCode != ETEXIT_CODE_SJIS)  return -4;

    peds2 = &pOutArg->data.edstring;
    for (i = 0; i < peds1->lLength; i++) {
         if (peds1->pszString[i] !=  ' ') break;
    }
    if (i < peds1->lLength){
        strncpy(peds2->pszString, peds1->pszString,
peds1->lLength);
        peds2->pszString[peds1->lLength] = '¥0';
    } else {
        strcpy(peds2->pszString, "(space)");
    }
    peds2->lLength = strlen(peds2->pszString);
    peds2->lCode = ETEXIT_CODE_SJIS;

    return 0;
}
```

● C 言語の出口関数の実装例（日付時刻型）

```
/*
 *  userdtm.c : 日付時刻型ユーザ関数の実装例
 */

#include  <stdlib.h>
#include  <string.h>
#include  "ETexit.h"

long  userdtm(long lArgNum, ETEXITARG *ppInArg[], ETEXITARG
*pOutArg)
{
    /* 特定の日時を設定する */
    pOutArg->data.eddtm.lYear = 2002;
    pOutArg->data.eddtm.lMonth = 2;
    pOutArg->data.eddtm.lDay = 15;
    pOutArg->data.eddtm.lHour = 12;
    pOutArg->data.eddtm.lMinute = 0;
```

```
    pOutArg->data.eddtm.dSecond = 0.0;
    strcpy(pOutArg->data.eddtm.pszZone, "+09:30");

    return 0;
}
```

● C 言語の出口関数実装例（実行環境文字コード文字列型と API からのポインタを使用したユーザ関数）

```
/*
 *  userPrintEnvstring.c : 実行環境文字コード文字列型とAPIからのポインタを使用したユーザ関数の実装例
 */
#include  <string.h>
#include  <stdio.h>
#include  "ETexit.h"

/* 実行環境文字コード型引数で受け取った文字列を，APIから渡されたファイルに書き込む関数 */
long  userPrintEnvstring(long lArgNum, ETEXITARG *ppInArg[],
                         ETEXITARG *pOutArg)
{
    ETEXITDATAENVSTRING *pEdEnvstr = NULL;
    ETEXITDATASTRING *peds = NULL;
    const char *cszFile = NULL;
    FILE *fp = NULL;

    if (lArgNum != 2 || ppInArg == NULL)  return -1;
    if (ppInArg[0] == NULL || ppInArg[0]->lType != ET_ENVSTRING)
            return -2;
   if (ppInArg[1] == NULL || ppInArg[1]->lType != ET_PTR)  return -3;

    pEdEnvstr = &ppInArg[0]->data.edenvstring;

   if (pEdEnvstr == NULL || pEdEnvstr->lErrCode == -1)  return -4;
    else {
          /* 実行環境の文字コードは SJISを仮定 */
          if (pEdEnvstr->lCode != ETEXIT_CODE_SJIS)  return -5;
    }

    /* APIから渡されたファイル名を取り出す */
    cszFile = (const char *)ppInArg[1]->data.pPtr;
    if (!cszFile)  return -6;

    /* ファイルに引数で受け取った文字列を書き込む */
    fp = fopen(cszFile, "w");
    if (!fp)  return -7;

    fprintf(fp, "%s", pEdEnvstr->pszString);
    if (fp)  fclose(fp);

    /* ユーザ組み込み関数の戻り値には引数の文字列をそのまま渡す */
    peds = &pOutArg->data.edstring;
    strcpy(peds->pszString, pEdEnvstr->pszString);
    peds->lLength = strlen(peds->pszString);
    peds->lCode = ETEXIT_CODE_SJIS;

    return 0;
}
```

11.4.2 ダイナミックリンクライブラリ（DLL）の作成（Windows の場合）

ユーザが作成した出口関数を含む DLL を作成して，トランスレータとバインドします。作成した DLL に未解決のシンボルがあると，変換実行時に DLL をロードしたときにエラーとなります。

DLL の作成手順を次に示します。

（1） 32bit 対応の DLL の作成

出口関数を含む 32bit 対応の DLL を作成します。DLL 名はシステム情報ファイル「ettrans.ini」内で指定したものと同じにします。また，出口関数は次のどちらかの方法で外部にエクスポートする必要があります。

- モジュール定義ファイル（.def）を作成し，出口関数名を記述する
- 出口関数のソースコード内で __declspec（export）宣言する

（2） 作成した DLL のコピー

作成した DLL ファイルをシステム情報ファイル「ettrans.ini」内で指定したパスにコピーします。

11.4.3 共用ライブラリの作成（ワークステーションの OS の場合）

ユーザが作成した出口関数を含む共用ライブラリを作成して，トランスレータとバインドします。作成した共用ライブラリに未解決のシンボルがあると，変換実行時に共用ライブラリをロードしたときにエラーとなります。

共用ライブラリの作成手順を次に示します。

（1） makefile の作成

Interschema が提供するサンプル makefile を参考にして，共用ライブラリを作成してください。サンプルファイル makefile の格納場所は，リリースノートを参照してください。

（2） 作成した共用ライブラリのコピー

作成した共用ライブラリをシステム情報ファイル「ettrans.ini」内で指定したパスにコピーします。

11.5 Java 言語の出口関数の作成

Java 言語の出口関数は，任意のクラス内の static メソッドとして作成します。

この節では，Java 言語の出口関数の作成方法について説明します。

11.5.1 Java 言語の出口関数の定義

ユーザ組み込み関数の機能を，Java 言語の出口関数中に記述します。

(1) Java 言語の出口関数の仕様

Java 言語の出口関数の仕様を次に示します。

(a) 引数

Java 言語の出口関数の第 1 引数には，java.lang.Object 型を指定する必要があります。第 2 引数以降には，ユーザ組み込み関数の引数のデータ型（Interschema の定義型）を Java 言語のデータ型に置き換えたデータ型を指定します。ユーザ組み込み関数の引数のデータ型と Java 言語のデータ型の対応を，次の表に示します。

表 11-4 ユーザ組み込み関数の引数のデータ型と Java 言語のデータ型の対応

ユーザ組み込み関数のデータ型（Interschema 定義型）	対応する Java 言語のデータ型	備考
ET_INT	int	signed 32Bit
ET_REAL	double	64Bit format IEEE754
ET_STRING(戻り値)	java.lang.String	―
ET_ENVSTRING(引数)	jp.co.Hitachi.soft.interschema2.exitfunc.ENVString ※ 1	―
ET_STREAM	byte []	―
ET_DTM	jp.co.Hitachi.soft.interschema2.exitfunc.DateTime ※ 2	―

（凡例）

―：特にありません。

注※ 1

jp.co.Hitachi.soft.interschema2.exitfunc.ENVString の定義については，「10.3.1 ENVString クラス」を参照してください。

注※ 2

jp.co.Hitachi.soft.interschema2.exitfunc.DateTime の定義については，「10.3.2

DateTime クラス」を参照してください。

例えば，ユーザ組み込み関数 ET_STRING FUNC099 (ET_INT, ET_INT) に対応する Java 出口関数は，public static String func099(Object o, int a, int b) というシグニチャを持ちます。

(b) 例外

Java 言語の出口関数内でエラーが発生した場合，呼び出し元に例外をスローすることでエラーを通知します。この場合，ユーザが ExitFuncException クラスのインスタンスに設定したエラーコードが，ログファイルに出力されて，変換は中止されます。また，DateTime オブジェクトの生成に失敗したときは，Interschema が 0x20000004 以上のエラーコードを設定して，ExitFuncException をスローします。ExitFuncException クラスについては，「10.3.4　ExitFuncException クラス」を参照してください。

Java 言語の出口関数内で警告が発生した場合，呼び出し元に ExitFuncWarning 例外をスローすることで警告を通知します。この場合，Interschema は，ユーザが ExitFuncWarning クラスのインスタンスに設定した値を，Java 言語の出口関数の戻り値として使用するため，ユーザが例外に設定したエラーコードがログファイルに出力された後，変換処理は続行されます。ExitFuncWarning クラスについては，「10.3.3 ExitFuncWarning クラス」を参照してください。

Java 言語の出口関数から，ExitFuncException クラス又は ExitFuncWarning クラス以外の例外をスローしないでください。

Java 言語の出口関数から，直接又は ExitFuncWarning 例外を通して，基本型以外の Java クラスインスタンスを返す場合の戻り値についての制限事項を，次の表に示します。

表 11-5　出口関数戻り値の制限

ユーザ組み込み関数戻り値型	Java 言語で対応する型	制限
ET_STRING	java.lang.String	null，空文字列は設定できません。
ET_STREAM	byte []	null は設定できません。配列サイズは，1 以上に設定する必要があります。
ET_DTM	jp.co.Hitachi.soft.interschema2.exitfunc.DateTime	null は設定できません。

(c) Java 言語の出口関数で使用できる Java クラス

Interschema が定義している出口関数で使用できる Java クラスは，jp.co.Hitachi.interschema2.exitfunc パッケージで定義されています。jp.co.Hitachi.interschema2.exitfunc パッケージで定義されている Java クラスについては，「10.1.4　jp.co.Hitachi.soft.interschema2.exitfunc パッケージのクラスの概要」を

参照してください。

（2） Java 言語の出口関数の実装例

Java 言語の出口関数の実装例を次に示します。

● Java 言語の出口関数の実装例

```
package usrpackagename;
import
jp.co.Hitachi.soft.interschema2.exitfunc.ExitFuncException;
/**
 * Javaによる出口関数を定義します。
 *
 */
public class UFunc {
        /**
         * 二つの整数値を受け取り，その剰余を返します。
         * この関数のettrans.iniの定義は以下のようになります。
         * ET_INT FUNC001 (ET_INT, ET_INT)
         * 第一引数のObjectは明示的には記述しません。
         * 関数名FUNC001は任意に付けた名前であり，Java
         * メソッド名と一致させる必要はありません。
         */
       public static int func001 throws ExitFuncException (Object
obj, int a, int b){
                if ( b == 0 )  throw new ExitFuncException(-1);
                return a % b;
         }
}
```

11.5.2 Java 言語の出口関数へのオブジェクト渡し

Java 言語の出口関数で実行時のコンテキストを判断するために，Translator.exec メソッド呼び出し時に指定した Object クラスのインスタンスを，Java 言語の出口関数に渡すことができます。

Java 言語の出口関数へのオブジェクトを渡す場合の記述例を次に示します。

● Java 言語の出口関数へのオブジェクト渡しの例

```
/* ユーザーが定義した任意のオブジェクト */
String  usrObj = new String(“Hello.”);
try{
        Translator trans = new Translator();
        MDLInfo mdlinfo = new MDLInfo(“Sample.mdl”);
        •••
        trans.exec(mdlinfo, sources, dests, usrObj);
        /* このデータ変換で呼び出された */
        /* 出口関数にusrObjが渡されます */
    }catch(TranslatorException e){
}
```

Java 言語の出口関数でオブジェクトを利用する場合の記述例を次に示します。

● 出口関数でのオブジェクト利用の例

```
●●●
public class UFunc {
        public static String printHello throws ExitFuncException
(Object obj){
        if ( obj == null )  throw new ExitFuncException(-1);
        System.out.println((String)obj);
        return (String)obj;
     }
}
```

11.5.3 Java 言語の出口関数使用上の注意

- マルチスレッドを使用してデータ変換 API（Java 言語）を呼び出す場合，Java 言語の出口関数は，再入可能である必要があります。Interschema は，指定された Java メソッドの呼び出しについて同期処理を行いません。
- システム情報ファイル「ettrans.ini」に指定したクラスは，データ変換処理 API（Java 言語）を呼び出す Java 環境のクラスパスに含まれている必要があります。
- ettrans コマンド及びデータ変換処理 API（C 言語）から Java 言語の出口関数を呼び出す変換処理を行う場合，初期化時にはエラーは報告されず，変換時にはエラーメッセージがログファイルに出力されて，変換が中止されます。システム情報ファイル「ettrans.ini」に上記以外の不正な記述がある場合，Interschema の動作は，「表 11-6　システム情報ファイル「ettrans.ini」の記述に不正がある場合の動作」のとおりです。初期化時に，表の項番の順に不正内容をチェックして，先に見付かった不正内容に対してエラー情報がログファイルに出力されます。

表 11-6 システム情報ファイル「ettrans.ini」の記述に不正がある場合の動作

不正内容	起動方法	出口関数の対応言語	Interschema の動作
[Userfunc_Mapfunc] セクションの記述に不正がある場合※	ettrans コマンド，又はデータ変換処理 API（C 言語）	C 言語	初期化時には，エラー情報をログファイルに出力します。データ変換に使用すると，エラー情報をログファイルに出力して，変換を中止します。
	データ変換処理 API（Java 言語）	C 言語，又は Java 言語	初期化時には，エラー情報をログファイルに出力します。データ変換に使用すると，エラー情報をログファイルに出力して，変換を中止します。
	FDL エディタ又は MDL エディタ	C 言語，又は Java 言語	エディタ起動時に，メッセージボックスで警告します。
[Userfunc_ExitfuncName] セクション，及び [Userfunc_JavaMethodName] セクションの両方に，同じ関数 ID を持つユーザ組み込み関数が定義されてる場合	ettrans コマンド，データ変換処理 API（C 言語），又はデータ変換処理 API（Java 言語）	－	初期化時には，エラー情報をログファイルに出力します。データ変換に使用すると，エラー情報をログファイルに出力して，変換を中止します。
	FDL エディタ又は MDL エディタ	－	エディタ起動時に，メッセージボックスで警告します。
[Userfunc_ExitfuncName] セクション，及び [Userfunc_JavaMethodName] セクションの両方に定義がない場合	ettrans コマンド，データ変換処理 API（C 言語），又はデータ変換処理 API（Java 言語）	－	初期化時には，エラー情報をログファイルに出力します。データ変換に使用すると，エラー情報をログファイルに出力して，変換を中止します。
	FDL エディタ又は MDL エディタ	－	エラーを無視します。

不正内容	起動方法	出口関数の対応言語	Interschema の動作
定義された出口関数が見付からない場合	ettrans コマンド，又はデータ変換処理 API（C 言語）	C 言語	初期化時には，エラー情報をログファイルに出力します。変換には，エラー情報をログファイルに出力して，変換を中止します。
	データ変換処理 API（Java 言語）	Java 言語	データ変換に使用すると，エラー情報をログファイルに出力して，変換を中止します。
	FDL エディタ又は MDL エディタ	C 言語，Java 言語	エラーを無視します。

（凡例）

－：該当しません。

注※

システム情報ファイル「ettrans.ini」の記述内容が不正となる条件は，ユーザ組み込み関数がバインドされている出口関数の言語（C 言語，又は Java 言語），又は [Userfunc_Option] セクション指定されるオプションによって異なります。

12 フォーマット作成時及びデータ変換時の注意事項

この章では，XML データの変換と各種フォーマットを使用したデータ変換について説明します。フォーマットの特徴，作成方法，及びデータ変換時の規則や注意事項について，フォーマットごとに説明します。

12.1 XML データ

この節では，Interschema で XML データを変換する場合の，XML データの対応について説明します。

12.1.1 DTD から MDL ツリーへの変換

MDL エディタを使用して DTD フォーマットを MDL へ挿入する場合，ELEMENT 定義の親子関係に従って，ツリーへ展開します。

DTD の ELEMENT 定義で，子タグを内容モデルに持つ ELEMENT は構造コンポーネント，テキスト（#PCDATA）だけを持つ ELEMENT は型コンポーネントとします。コンポーネント名はタグ名です。EMPTY 要素も型コンポーネントとして展開します。

内容モデル内に名前を持たない構造の定義がある場合，システムが自動的に名称を作成して，構造コンポーネントを作成します。このような，DTD 上で名前を持たない（タグがない）構造を，ダミー構造と呼びます。混在内容もダミー構造の一つです。

ダミー構造の名称は，親のコンポーネント名称（ローカル名）に「#n」（n は通し番号）を付けたものです。これはシステムが与える特殊な名称なので，名称の前後にシングルクォーテーションが付きます。

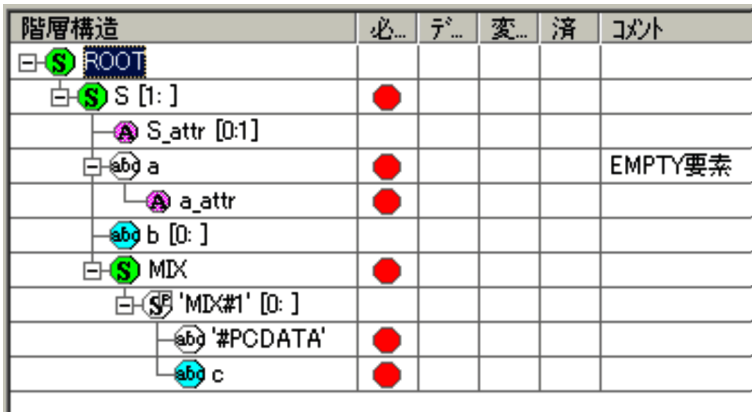
混在内容のテキスト要素に対する型コンポーネントの名前は「#PCDATA」となります（これもシステムが与える特殊な名称です）。

EMPTY 要素のコンポーネント，ダミー構造コンポーネント，混在内容のテキスト要素コンポーネントは，一般のコンポーネントとは異なる色（白）のアイコンで表示します。

DTD ファイルの記述例：

```
<!ELEMENT ROOT (S+)>
<!ELEMENT S (a,b*,MIX)>
<!ELEMENT a EMPTY>
<!ELEMENT b (#PCDATA)>
<!ELEMENT MIX (#PCDATA|c)*>
<!ELEMENT c (#PCDATA)>
<!ATTLIST S S_attr CDATA #IMPLIED>
<!ATTLIST a a_attr CDATA #REQUIRED>
```

DTD フォーマット挿入後のツリー表示：



12.1.2　XML 属性の扱い

XML 属性は，フォーマットツリー上で構造／型コンポーネントの下のアイテムとして表示されます。1 属性が 1 アイテムとして扱われ，マッピングの対象になります。また，構造コンポーネントの XML 属性は，この構造の子コンポーネントより上部に表示します（「12.1.1　DTD から MDL ツリーへの変換」の DTD ファイルの記述例での「S_attr」コンポーネント）。

デフォルト値を持つタイプの属性は，一般の型コンポーネントでデフォルト式を定義したものと同様で，マップ式が定義されていない場合は，デフォルト値が出力されます。

ただし，属性のデフォルト値タイプが「#FIXED+ デフォルト値」である場合は，システムが「=%UNDEF;」のマップ式を与え，そのままの状態では属性値が出力されないようにしています。この属性値を出力する場合は，マップ式を削除してください。

また，出力する「列挙型」の XML 属性に，属性の値として定義されていない値が代入されたり，「#FIXED+ デフォルト値」の XML 属性にデフォルト値と異なる値が代入されたりした場合は，変換実行時にエラーとなるので注意してください。

XML 属性のグローバル名は，属性名の前を「@@」でつなぎます。その他のコンポーネント部分は，一般のコンポーネントと同じように「@」でつなぎます。

「12.1.1　DTD から MDL ツリーへの変換」の DTD ファイルの記述例で，コンポーネント「a」の属性「a_attr」のグローバル名は次のようになります。

```
Format@ROOT@S@a@@a_attr
```

トランスレータが XML 属性のマップ式を評価するタイミングは，XML 属性が属する構造，又は型コンポーネントのマップ式を評価した後です。XML 属性が属するコンポーネントのマップ式に評価順序遅延指定がある場合，XML 属性の評価も同時に遅延されます。XML 属性自身のマップ式では，評価順序遅延の指定はできません。

一つのコンポーネントに複数の XML 属性がある場合は，MDL エディタのツリー表示で上部にあるものから順番に評価されます。

また，XML 属性のマップ式内で使用する階層指定（「#」「¥n」や INDEX 関数のパラメタ）は，XML 属性が属するコンポーネントを基点とします。XML 属性自身は階層指定外となります。

「12.1.1　DTD から MDL ツリーへの変換」の DTD ファイルの記述例で，コンポーネント「a」の属性「a_attr」のマップ式を例にとると，次のようになります。

```
#          ：「a」を表す
¥1         ：「S」を表す
INDEX(1)：「S」の現在のインデクスを表す
```

12.1.3　EMPTY 要素の扱い

EMPTY 要素とは，要素の内容を持っていない空要素です。

DTD で EMPTY 要素として定義されたコンポーネントでは，EMPTY 要素タグの中のすべてのデータは，無視されます。ただし，XML 属性は独立したコンポーネントなので，無視されません。EMPTY 要素は，次のような場合に使用します。

例えば，トランスレータへ入力するデータは変更できないで，変換に使用しない DTD が一意に決まらない XML フォーマットがあったとします。この場合，必要としない最上位の ELEMENT を EMPTY 要素として定義した DTD を作成して，その DTD を MDL に取り込みます。容易にデータを変換できます。

12.1.4　ANY 要素の扱い

ANY 要素とは，テキストの要素，及び宣言済みの要素が任意に出現できるようにする任意要素です。

入力フォーマットの ANY 要素として定義されたコンポーネントに対応する XML データは，文字列化されて，型コンポーネントのデータとなります。文字列化されたデータは，基となった XML データと同等の文字列です。文字列化の対象となるのは，要素タグ，XML 属性，及びテキストデータです。テキストデータに「<」などの特殊な文字が含まれる場合は，該当する文字は文字参照に置換されます。

出力フォーマットの ANY 要素である型コンポーネントに対して値がマッピングされている場合は，マッピングされた値が ANY 要素のテキストデータとして出力されます。

また，次の場合は ELEMENT が強制的に ANY 要素として取り込まれます。

- 再帰構造の展開ネスト数 =1 での展開後のコンポーネント数が最大値を超えた場合
  DTD 挿入コマンドを実行して，再帰コンポーネントを ANY 要素として展開するかど

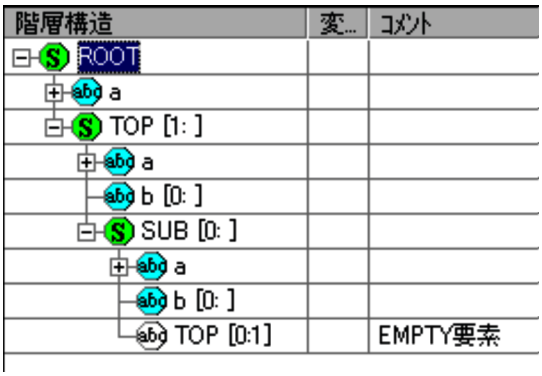
うかを問い合わせるダイアログで「はい」を選択したとき，ルートからの再帰を除く再帰の最上位構造が ANY 要素として取り込まれます。

- 混在内容構造を ANY 要素として取り込むように指定した場合
  オプションコマンドで「混在内容構造を ANY 型にする」を指定したとき，混在内容構造が ANY 要素として取り込まれます。混在内容構造については，「12.1.6　混在内容構造の扱い」を参照してください。

12.1.5　再帰構造を含む DTD 及び XML 文書の扱い

再帰構造が任意のネスト数を持つ XML データは変換できません。再帰構造を持つ DTD を挿入する場合は，決められたネスト数分を，ツリーとして展開してから扱います。MDL エディタでは，指定されたネスト数を超える部分（再帰構造の末端）はダミーの EMPTY 要素として表示し，マッピングの対象外とします。また，MDL で展開されたネスト数を超える入力 XML データがトランスレータに渡された場合，超過部分のデータは無視されます。

展開ネスト数に 1 を指定した例を示します。図の構造「TOP」が再帰構造です。



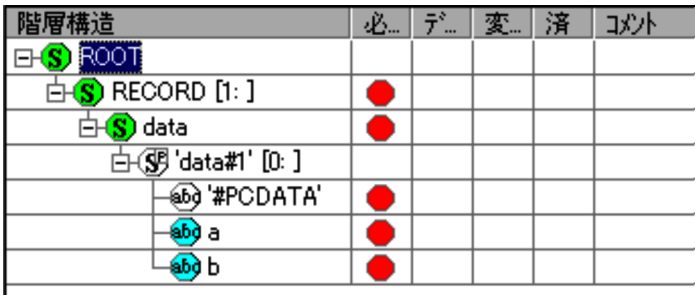
挿入する DTD フォーマットに再帰構造が含まれる場合，［展開ネスト数指定］ダイアログが表示されます。このダイアログで，ツリーへ展開するネスト数を指定してください。

なお，展開ネスト数が 1 の状態でコンポーネント数が再帰展開最大コンポーネント数を超えると，再帰の最上位構造を ANY 要素にするか確認するダイアログが表示されます。ダイアログで「はい」を選択した場合，再帰の最上位構造が ANY 要素として取り込まれます。ただし，再帰構造の最上位がルート場合は，ANY 要素として取り込まれません。再帰展開最大コンポーネント数は，オプションコマンドで設定できます。ただし，DTD の挿入（DTD マージ含む）では，この最大数を超えるコンポーネントを作成することはできません。

12.1.6　混在内容構造の扱い

混在内容構造とは，テキストの要素，及び宣言済みの要素が混在する構造です。図 12-1 では，「data」が混在内容構造になります。

図 12-1　混在内容構造の表示例



混在内容構造は再帰構造となることが多く，DTD を挿入する場合にコンポーネント数が最大値を超えてしまうことがあります。このような場合，次の指定をすることで，混在内容構造を ANY 要素として取り込むことができます。

1. MDL **エディタのメニューから［ツール］－［オプション］を選択します。**
   ［オプション］ダイアログが表示されます。
2. **［全般］タグの「混在内容構造を ANY 型にする」を指定します。**

なお，MDL エディタがコンポーネントを混在内容構造と見なす条件は次のとおりです。

- 子コンポーネントが選択構造のダミー構造だけ，かつ出現数が 0 回以上（最大指定なし）
- 子コンポーネント（ダミー構造）が複数の子を持ち，かつ先頭の子が #PCDATA

12.1.7　XML データの変換

XML データを変換する場合，XML 文書のエンコーディングに関係なく，一度 UTF-16 へ文字コード変換してからデータ変換されます。

12.1.8　未サポート項目

未サポートの項目を次に示します。これらが定義されている DTD ファイルを MDL へ取り込む場合，DTD フォーマット挿入実行時にエラーとなります。

- タグ名，属性名，属性値（デフォルト値，列挙値）としてシフト JIS 表記できない文字を含むもの

12.2 CSV フォーマット

この節では，CSV フォーマットの特徴，フォーマット作成時の注意事項，及び CSV フォーマットを使用したデータ変換の規則について説明します。

12.2.1 CSV フォーマットの特徴

FDL ファイルを作成するときに，フォーマットの規格を「CSV」と指定すると，変換データを CSV 形式で処理できるようになります。フォーマットの規格が「CSV」となっているフォーマットを，CSV フォーマットと呼びます。CSV フォーマットでは，ダブルクォーテーション「"」で囲まれた部分がデータとして扱われます。データは，型コンポーネントごとに処理されます。

12.2.2 CSV フォーマットの作成

CSV フォーマットを作成するときの注意事項について説明します。

CSV フォーマットでは，型のサイズを可変長にして，不要文字削除指定を設定しておく必要があります。文字コードには初期値（JIS8，SJIS）を，セパレータには予約セパレータ（#Comma 又は #Linefeed）を利用することをお勧めします。また，原則として，次に示す型を使用してください。

- 1 バイト文字列型
- 混在文字列型
- 整数型
- 実数型
- 暗黙的小数部付数値型
- 日付型
- 時刻型
- 日付時刻型

CSV フォーマットでは，数値型の小数点及び指数記号に「"」を使用したり，桁セパレータを定義したりしないでください。

次に，FDL ファイルの検証時に，エラー又はワーニングエラーとなる場合について説明します。

エラーとなる場合

- データに解放文字（「?」など）が含まれていると，FDL ファイルの検証時にエラーとなります。CSV フォーマットでは解放文字を使用しないようにしてください。

ワーニングエラーとなる場合

- 型の文字コードとセパレータの文字コードが一致しないと，フォーマットの検証

時にワーニングエラーとなります。
- フォーマットに文字列型数値以外の数値型，又はバイト列型が含まれていると，フォーマットの検証時にワーニングエラーとなります。文字列型数値以外の数値型（符号付 2 進整数など），及びバイト列型は使用しないようにしてください。

12.2.3 CSV フォーマットの変換の規則

CSV フォーマットでのデータ処理の規則について説明します。

- データ入力時にダブルクォーテーションで囲まれている箇所がある場合は，ダブルクォーテーションで囲まれている内部をデータとして扱います。データ中の「" "」は，「"」として扱います。例えば「"x,y," "z,"」というデータは，「x,y," z,」として扱います。
- 出力するデータにコンマが含まれる場合，又はデータの先頭がダブルクォーテーションの場合は，出力データがダブルクォーテーションで囲まれます。このとき，データ中のダブルクォーテーションは，ネストされます。
- データに対するダブルクォーテーション囲み処理は，型コンポーネントごとに実行されます。
- 実行時オプション「-CSV」を指定して CSV フォーマットを出力する場合，データは常にダブルクォーテーションで囲まれます。ただし，バイト列などは囲まれません。
- 囲み文字としてのダブルクォーテーションは，データサイズに含まれません。ただし，囲み文字によってデータ中にネストされたダブルクォーテーションは，文字数には含まれませんが，バイト数には含まれます。
- バイト列系の型と 2 バイト文字列は，CSV 処理の対象になりません。

12.3　CII3 フォーマット

この節では，CII3 フォーマットの特徴，フォーマット作成方法，及び CII3 フォーマットを使用したデータ変換の規則について説明します。

12.3.1　CII3 フォーマットの特徴

CII3 フォーマットは，CII 標準対応のフォーマットです。CII1.51，CII2.10，及び CII3.00 に対応していて，電子データ交換ツールでサポートしている従来の CII フォーマットよりも，効率良くデータを変換できます。CII 標準データを変換する場合は，CII3 フォーマットを使用することをお勧めします。

CII3 フォーマットは，縮小と拡張を区別しません。また，TFD を一つの型コンポーネントとして扱うため，フォーマットの定義やマッピングが容易にできます。

12.3.2　CII3 フォーマットの作成

CII3 フォーマットは，Interschema が提供しているフォーマットを基に作成してください。必要な TFD 型，メッセージ構造，及びマルチ明細構造を定義して，Interschema が提供しているフォーマットのツリー構造に追加します。作成方法の詳細については，FDL エディタ及び MDL エディタのヘルプを参照してください。

12.3.3　CII3 フォーマットの変換の規則

CII3 フォーマットでのデータ処理の規則について説明します。

（1）データの入力処理の規則

CII3 フォーマットでのデータの入力処理の規則を次に示します。

- データ中にマルチ明細構造がある場合，マルチ明細の中の TFD で最初に出現したデータのタグ番号と一致する番号の TFD 型コンポーネントを子に持つマルチ明細コンポーネントが対応付けられます。
- MDL で最小出現回数が 1 と定義された TFD 及びマルチ明細の子に対応するデータが入力されなかった場合，トランスレータはコンポーネントの値をスペース（K 属性及び X 属性），0（9 属性，N 属性，及び Y 属性），0x00（B 属性）だと仮定して処理します。
- 入力データのエラーの原因が構造コンポーネントにある場合，ログファイルのオフセットには次に示す上位構造の先頭位置が出力されます。
  - メッセージグループ・ヘッダ
  - メッセージ
  - バイナリ

- メッセージグループ・トレーラ
- 同報ヘッダ

- 標準以外の文字コードの CII データを変換する場合，使用する文字コードをフォーマットの文字コードとして定義しておく必要があります。

(2) データの出力処理の規則

CII3 フォーマットでのデータの出力処理の規則を次に示します。

- 省略できる TFD は出力されません。
- TFD を省略したことによってマルチ明細を省略する場合，縮小モードのときはマルチ明細のヘッダとトレーラだけが出力されます。拡張モードのときは，何も出力されません。
- CII のバージョンは，メッセージグループ・ヘッダの子で決定します。メッセージグループ・ヘッダの子でシンタックスのバージョンを表す型コンポーネントにマッピングした値が「CII151」の場合は CII1.51 形式，「CII210」の場合は CII2.10 形式，「CII300」の場合は CII3.00 形式になります。
- TFD に対する出力データの文字コードは，フォーマットの文字コードに従います。初期値の文字コード（JIS8，JISK）を設定することをお勧めします。ただし，K 属性を Unicode で出力した場合は，メッセージグループ・ヘッダの子コンポーネントで 2 バイト文字コードを表す型コンポーネントに，「U」をマッピングする必要があります。
- 縮小／拡張モードは，メッセージグループ・ヘッダの子コンポーネントで拡張モードを表す型コンポーネントにマッピングした値に従って決定されます。ただし，CII3.00 形式の場合は，常に拡張モードになります。
  拡張モードを表す型コンポーネントにマッピングした値が，「S」又はスペースの場合は縮小モード，「E」の場合は拡張モードになります。
- 分割（分割固定長）／通常（分割可変長）モードは，メッセージグループ・ヘッダの子コンポーネントで分割モードを表す型コンポーネントにマッピングした値に従って決定されます。
  分割モードを表す型コンポーネントにマッピングした値が，「S」の場合は通常（分割可変長）モード，「M」又はスペースの場合は分割（分割固定長）モードになります。
- CII1.51 形式でハッシュトータル値を設定したい場合は，メッセージグループ・ヘッダの子コンポーネントでトータル項目番号を表す型コンポーネントに，該当するタグ番号をマッピングします。また，EIAJ のハッシュ計算方法で値を出力した場合は，実行時オプション「-EIAJHASH」を指定して，変換します。
- 標準以外の文字コードの CII データを変換する場合，使用する文字コードをフォーマットの文字コードとして定義しておく必要があります。出力フォーマット上のメッセージグループヘッダ下の対応コンポーネントは，マッピングしなくてもかまいません。

12.4　EDIFACT4 フォーマット

この節では，EDIFACT4 フォーマットの特徴，フォーマット作成方法，及び EDIFACT4 フォーマットを使用したデータ変換の規則について説明します。

12.4.1　EDIFACT4 フォーマットの特徴

EDIFACT4 フォーマットは，UN/EDIFACT 対応のフォーマットです。バッチ形式構造に対応していて，電子データ交換ツールでサポートしている従来の EDIFACT フォーマットよりも，効率良くデータを変換できます。EDIFACT データを変換する場合は，EDIFACT4 フォーマットを使用することをお勧めします。

EDIFACT4 フォーマットは，一つのフォーマットで UNOA や UNOB などのシンタックス識別，及び EDIFACT のバージョンに対応しています。フォーマットでセパレータ値を個別に指定する必要はありません。

なお，EDIFACT の Version.4 と Version.1 ～ 3 との間には，サービスセグメント関連に違いがあります。EDIFACT の Version.4 と Version.1 ～ 3 のサービスセグメント関連については，「付録 H.1　対応するバージョン」を参照してください。

12.4.2　EDIFACT4 フォーマットの作成

EDIFACT4 フォーマットは，Interschema が提供しているフォーマットを基に作成してください。必要なメッセージ構造，セグメント，複合要素，繰り返し構造，及び型を定義して，Interschema が提供しているフォーマットのツリー構造に追加します。

Interschema が提供するフォーマットには，EDIFACT の Version.4 用と，Version.1 ～ 3 用の 2 種類があります。EDIFACT の Version.4 用の入力フォーマットには，Version.1 ～ 3 のデータも入力できますが，あらかじめ変換するデータの EDIFACT のバージョンが分かっている場合は，対応するバージョンのフォーマットを使用してください。

作成方法の詳細については，FDL エディタ及び MDL エディタのヘルプを参照してください。

12.4.3　EDIFACT4 フォーマットの変換の規則

EDIFACT4 でのデータ処理の規則について説明します。

- 入力データのエラーの原因が構造コンポーネントにある場合（コンポーネント出現回数不正など），ログファイルのオフセットには上位のセグメント構造の先頭位置が出力されます。エラーがセグメント構造のものの場合は，自身のオフセットが出力されます。
- データの出力時には，シンタックスのバージョンを表す型コンポーネントにマッピン

グした値が，出力データの EDIFACT のバージョンになります。

12.5　Sea-NACCS フォーマット

この節では，Sea-NACCS フォーマットの特徴，フォーマット作成方法，及び Sea-NACCS フォーマットを使用したデータ変換の規則について説明します。

12.5.1　Sea-NACCS フォーマットの特徴

Sea-NACCS フォーマットには，Sea-NACCS EDI 電文フォーマット，Sea-NACCS EDIFACT 電文フォーマットなどがあります。

Interschema では，Sea-NACCS のシステム運用開始時の規定に応じて，Sea-NACCS フォーマットを異なる規格種別のフォーマットとして扱います。

Sea-NACCS のシステム運用開始時と Interschema の規格種別の対応を次の表に示します。

表 12-1　Sea-NACCS のシステム運用開始時と Interschema の規格種別の対応

Sea-NACCS のシステム運用開始時	Interschema の規格種別	意味
1999 年 10 月	SNACCS	Sea-NACCS EDI 電文フォーマット（旧）
	SNEDIFACT	Sea-NACCS EDIFACT 電文フォーマット（旧）
2008 年 10 月	SNACCS2	Sea-NACCS EDI 電文フォーマット
	SNEDIFACT2	Sea-NACCS EDIFACT 電文フォーマット

1999 年 10 月に運用が開始された Sea-NACCS 用のフォーマットは，電子データ交換ツールの海上貨物通関用辞書を使用することを前提としています。

2008 年 10 月の Sea-NACCS システム更改後のフォーマットは，Interschema がサンプルとして提供するフォーマットをカスタマイズして使用することを前提としています。

12.5.2　Sea-NACCS フォーマットの作成

ここでは，SNACCS2 ／ SNEDIFACT2 フォーマットの作成方法について説明します。

SNACCS2 ／ SNEDIFACT2 フォーマットは，Interschema がサンプルとして提供するフォーマットを基に作成してください。

Sea-NACCS EDI 電文フォーマットの場合は，必要な業務の項目（型）を追加定義し，業務メッセージ構造を作成します。Sea-NACCS EDIFACT 電文フォーマットの場合は，必要な業務に対応するメッセージから不要なコンポーネントを削除して作成します。

作成方法の詳細については，FDL エディタ及び MDL エディタのヘルプを参照してください。

12.5.3 Sea-NACCS フォーマットの変換の規則

Sea-NACCS フォーマットでのデータ処理の規則について説明します。なお，「付録 J Sea-NACCS のサポート範囲」も合わせて参照してください。

Interschema では，処理要求電文は出力変換を想定しています。また，処理結果通知電文，及び出力情報電文は，入力変換を想定しています。

（1） Sea-NACCS EDI 電文の入力処理の規則

MDL で定義されたコンポーネントのデータが現れる前に入力データが終了した場合，スペース項目があるものとして処理します。

（2） Sea-NACCS EDI 電文の出力処理の規則

Sea-NACCS フォーマットでの EDI 電文の出力処理の規則を次に示します。

- 実行時オプション「-SNCODE」が指定されている場合，Sea-NACCS フォーマットに対する文字コードチェックを実施します。
- NULL マップ式が定義された要素に対して，スペースを出力します。
- コンポーネントのマップ式が評価できない場合，デフォルト式の値を出力します。
- 出力データ末尾の不要なスペース項目をカットします。

（3） Sea-NACCS EDIFACT 電文の規則

Sea-NACCS フォーマットでの EDIFACT 電文の規則を次に示します。なお，次に示す規則以外は，一般の EDIFACT フォーマットの変換と同じです。

- テストモードは判定しません。
- 出力変換（処理要求電文を出力する変換）で，実行時オプション「-SNCODE」が指定されている場合，Sea-NACCS フォーマットに対する文字コードチェックを実施します。

（4） Sea-NACCS フォーマットに対する文字コードチェックの規則

Sea-NACCS フォーマットに対する文字コードチェックの規則を次に示します。

- 出力変換で，かつ実行時オプション「-SNCODE」が指定されている場合に，Sea-NACCS フォーマットに対する文字コードチェックを実施します。
- Sea-NACCS 文字セットの最大範囲でチェックします。なお，業務によって使用できる文字と使用できない文字がありますが，Interschema では常に使用できる文字として扱います。

付録

付録 A　トラブルシューティング

トランスレータでエラーが発生したときの処理について説明します。

付録 A.1　エラー発生時の処理

トランスレータでエラーが発生したときの処理，及びエラー発生時の対策としてログファイルの参照方法について説明します。

(1) FDL 及び MDL エディタ

エディタ画面上にエラーメッセージを出力します。メッセージに従って対処してください。

(2) トランスレータ

ここでは，コマンド又は API でトランスレータを実行している場合にエラーが発生したときの処理を説明します。データ変換処理 API（Java 言語）を使用している場合，エラー発生時には，固有の処理があります。データ変換処理 API（Java 言語）を使用している場合は，ここでの説明と合わせて，「付録 A.1(3)　データ変換処理 API（Java 言語）での実行時に固有の処理」を参照してください。

(a) エラー発生時の処理

トランスレータは，エラー発生時に次のように処理します。

- 戻り値をログファイルに出力
- 変換時のエラー情報をログファイルに出力
- API での実行時に，エラー情報をエラー情報リスト（API パラメタ）として出力
- -CIIE オプション指定時に CII 標準形式ファイルを入力した場合，CII エラー情報メッセージファイルを出力
- 例外によってエラーを通知（データ変換処理 API（Java 言語）での実行時）
  データ変換処理 API（Java 言語）での実行時のエラーについては，「付録 A.1(3)　データ変換処理 API（Java 言語）での実行時に固有の処理」を参照してください。

各処理について次に説明します。

戻り値をログファイルに出力

戻り値をログファイルに出力します。メッセージに戻り値を付けて出力するので，メッセージのエラーレベルを判定できます。エラーレベルについては，「付録 B.1　メッセージ形式」を参照してください。

変換時のエラー情報をログファイルに出力

変換時のエラー情報としてエラーメッセージをログファイルに出力します。ログファイルを参照して，エラーの詳細情報を確認できます。メッセージについては「付録 B.3　メッセージ」を参照してください。

API での実行時に，エラー情報をエラーリスト（API パラメタ）に出力

API での実行時に，エラー情報をエラーリストに出力します。
C 言語の API 及びエラーリストについては「9.2.1　ETtrans2Init（トランスレータの初期化）」，及び「9.2.9　ETtrans2Exec（変換実行）」を参照してください。Java 言語の API のエラー情報については，「付録 A.1(3)(b)　エラーコード」を参照してください。

CII 標準形式ファイルを入力した場合に，CII エラーメッセージファイルを出力

-CIIE オプション指定時に CII 標準形式ファイルを入力した場合，CII エラーメッセージファイルを出力します。メッセージ内に設定するエラーコードと，トランスレータで出力するエラーメッセージとの対応については，「付録 G　CII シンタックスルールのサポート範囲」及び「付録 G.3　エラーコードの対応」を参照してください。

(b) ログファイルの参照方法

トランスレータのログファイルは，データの変換結果として戻り値やエラーメッセージなどを出力するテキストファイルです。ログファイルに出力する項目やログファイルの出力先について次に説明します。

ログファイルに出力する項目

- 変換日時
- プロセス ID（ワークステーションの OS だけ）
- 実行時のスレッド ID
- 戻り値
- エラーメッセージ
- エラーメッセージの付加情報
  エラーメッセージの付加情報には，次のような情報があります。
  - 変換フォーマット名，入出力データファイル名又はアドレス
  - 変換エラーとなった要素のグローバル名，及びエラーデータのオフセット値
  - エラーが発生した出口関数戻り値，及びユーザ組み込み関数名
  - 指定不正オプション，及びフォーマット名

  など

ログファイルの出力先と参照方法

ログファイルは，コマンド実行時に指定したファイル（省略時はデフォルトログファイル）へ出力されるので，テキストエディタなどで開いて参照できます。

ログファイル名

ログファイル名は，トランスレータのコマンドの引数として指定できます。ログファイル名を指定しなかった場合は，デフォルトのログファイル「errlog.txt」（*Interschema のインストールディレクトリ*¥log 下に格納）にログが出力されます。

また，ログファイル名を指定しても，ファイル名に誤りがあるなどの理由でファイルを開けなかった場合には，デフォルトのログファイル，又は標準出力にログが出力されます。

ログファイルのサイズ

ログファイルは，システム情報ファイル「ettrans.ini」に指定した最大サイズ以内で出力されます。指定した最大サイズを超えた場合は，ファイル内の古い出力情報から削除されます。ログファイルの初期値の最大サイズは 1,000 キロバイトです。ただし，ログ出力抑止指定がある場合は出力しません。最大サイズは，1 キロバイトから 10,000 キロバイトまでの範囲で変更できます。最大サイズを変更する場合は，「5.3(1)(a)　ログファイルサイズの変更」を参照してください。ログファイルの出力を抑止する場合は，「5.4.3　システム情報ファイル「ettrans.ini」での実行時オプションの定義」を参照してください。

(3) データ変換処理 API（Java 言語）での実行時に固有の処理

ここでは，データ変換処理 API（Java 言語）でトランスレータを実行している場合に固有のエラーが発生したときの処理について説明します。データ変換処理 API（Java 言語）でトランスレータを実行している場合にエラーが発生した場合，「付録 A.1(2)　トランスレータ」の内容と合わせて，ここで説明する内容が必要になります。

(a) エラー発生時の処理

データ変換処理 API（Java 言語）は，エラー発生時に次のように処理します。

- エラー情報をログファイルに出力
- 例外によってエラーを通知

データ変換処理 API（Java 言語）は，Java Navive Interface（JNI）を使用してデータ変換処理 API（C 言語）を呼び出します。データ変換処理 API（C 言語）の処理でエラーが発生した場合は，エラー情報をログファイルに出力します。データ変換処理 API（Java 言語）の処理でエラーが発生した場合は，例外によってエラーを通知します。

エラー情報をログファイルに出力

データ変換処理 API（C 言語）がログファイルに出力するエラー情報は，Translator クラスに指定する変換用のエラー情報，及びそれ以外のエラー情報があります。変換用のエラー情報以外のエラー情報は，デフォルトのログファイル（*Interschema のインストールディレクトリ*/log/errlog.txt）に出力されます。ログファイルに出力されるエラー情報については，「付録 B.3　メッセージ」を参照してください。

エラー情報を例外で通知

データ変換処理 API（Java 言語）が出力する例外については，「付録 B.4　データ変換処理 API（Java 言語）の例外」を参照してください。

(b) エラーコード

データ変換処理 API（Java 言語）で発生するエラーコードを次に示します。エラーコードの値は，TranslatorException 例外に設定されて，getErrorCode メソッドで取得できます。

表 A-1　エラーコード一覧

値		意味
(0x)2000xxxx		続行できないレベルのエラーが発生しました。
(0x)4000xxxx		トランスレータ内部で致命的なエラーが発生しました。
xxxx の値	0nnn※	各クラス共通のエラーが発生しました。
	1nnn※	初期化処理でエラーが発生しました。
	2nnn※	Translator クラスでエラーが発生しました。
	3nnn※	MDLInfo クラスでエラーが発生しました。
	4nnn※	DLProperty クラスでエラーが発生しました。
	5nnn※	TranslateData クラス，又はその派生クラスでエラーが発生しました。

注※

nnn のコードは，000 から順に付けられます。

付録 B　トランスレータのメッセージ

トランスレータのメッセージ形式，メッセージの見方，及びメッセージ内容について説明します。

付録 B.1　メッセージ形式

メッセージは次の形式で出力します。

KBETnnnny-l メッセージテキスト

KBETnnnny-l はメッセージを識別するためのメッセージ ID です。

nnnn：メッセージ番号

y：メッセージを出力したプログラム

- F：FDL エディタ
- M：MDL エディタ
- T：トランスレータ
- J：データ変換処理 API（Java 言語）

l：メッセージのエラーレベル

エラーレベルの意味と対応するトランスレータの戻り値を次に示します。

表 B-1　エラーレベル

エラーレベル	意味	対応するトランスレータの戻り値
I	インフォメーション	0x00ZZZZZZ
W	ワーニング	0x01ZZZZZZ
E	エラー	0x02ZZZZZZ，0x2000ZZZZ（データ変換処理 API（Java 言語）の場合）
S	システムエラー	0x04ZZZZZZ，0x4000ZZZZ（データ変換処理 API（Java 言語）の場合）

付録 B.2　メッセージの見方

メッセージは，次の形式で説明します。

メッセージ ID

メッセージテキスト

メッセージの内容

システムの処理

メッセージをログファイルに出力した後の主な処理内容

要因

想定されるエラー要因

対策

メッセージに応じた対策方法

付録 B.3　メッセージ

エラー発生時に，ログファイルに出力されるエラーメッセージについて説明します。

KBET0001T-I

次の MDL で変換を開始します。

データの変換前にログファイルに出力されるメッセージです。MDL ファイル名がログファイルに出力されます。

システムの処理

処理を続行します。

KBET0002T-I

入力データ

変換するデータを入力するときにログファイルに出力されるメッセージです。変換するデータのフォーマット名と，入力ファイル名又はメモリアドレスがログファイルに出力されます。

システムの処理

処理を続行します。

KBET0003T-I

出力データ

変換するデータを出力するときにログファイルに出力されるメッセージです。変換するデータのフォーマット名と，出力ファイル名又はメモリアドレスがログファイルに出力されます。

システムの処理

処理を続行します。

KBET0004T-I

変換処理が終了しました。

データの変換処理が終了したときにログファイルに出力されるメッセージです。メッセージに対応する戻り値がログファイルに出力されます。戻り値については，「5.4.2 ettrans コマンドの戻り値」を参照してください。

システムの処理

処理を終了して，トランスレータを終了します。

KBET0006T-I

1 グループ単位の変換処理が終了しました。

1 グループ単位のデータの変換処理が終了したときにログファイルに出力されるメッセージです。

システムの処理

処理を続行します。

KBET0007T-W

不正な文字コードをスペースへ置換して変換しました。

データ中の不正な文字コードをスペースに置き換えたときにログファイルに出力されるメッセージです。メッセージに対応するコンポーネント名又はオフセットがログファイルに出力されます。

システムの処理

処理を続行します。

要因

入力側と出力側で文字コードが異なり，かつ出力側で使用できる文字範囲が異なることが考えられます。

対策

データ中の不正な文字コードをスペースに置き換えたことによって，データ項目として不都合がないか確認してください。

KBET0008T-W

要素のサイズ／桁数があふれたためカットしました。

出力データのサイズが要素のサイズを超えたため，桁あふれ部分が切り捨てられたときにログファイルに出力されるメッセージです。メッセージに対応するコンポーネント名及びオフセットがログファイルに出力されます。

システムの処理

処理を続行します。

要因

出力データのサイズが型定義した要素のサイズを超えました。サイズ／桁数の桁あふれ部分の内容を次に示します。

表 B-2　サイズ／桁数の桁あふれ部分の内容の説明

項目	説明
サイズ／桁数 桁あふれ部分	• 出力要素が固定長で，データサイズが想定値を超えました。 • 小数部を持つ数値を，小数部を持たない数値に代入しようとしました。 • マップ式の演算結果が，想定する数値桁数を超えました。 • 出力側文字列中にセパレータと同じ文字があったため，解放文字を埋め込んだ分だけサイズが増加しました。 • 出力側文字列がシフトコードを使用するため，シフトコード分のサイズが増加しました。

対策

桁あふれ部分が切り捨てられたことによって，データ項目として不都合がないか確認してください。

KBET0009T-W

変換エラーが発生しましたが，正常処理したグループのみ出力しました。

システムの処理

処理を終了します。

対策

エラーを修正後，変換できたグループ以降を再度変換してください。

KBET0010T-I

入力データはテスト用のデータです。

入力したデータが，テスト用のデータであるときにログファイルに出力されるメッセージです。

システムの処理

処理を続行します。

KBET0011T-W

ハッシュトータル値がメッセージグループ・トレーラの値と一致しません。

CII のハッシュトータル値がメッセージグループ・トレーラの値と一致しないときにログファイルに出力されるメッセージです。メッセージに対応するタグ番号が出力されます。

システムの処理

処理を続行します。

要因

入力データが EIAJ シンタックスルール方式で加算した，対象とする TFD 項目の属性で不要な「0」があることが考えられます。

対策

ログファイルに出力されたデータタグ番号のデータ値を確認し，必要に応じてハッシュトータル値の加算方法を変更してください。

KBET0012T-W

出口関数の定義に不正があります。

ユーザ組み込み関数の定義に誤りがある場合にログファイルに出力されるメッセージです。メッセージに対応するシステム情報ファイル「ettrans.ini」の不正位置が，ログファイルに出力されます。

出口ライブラリのローディングの失敗，及び未解決シンボルによる不正時は，その詳細情報がログファイルに出力されます。

システムの処理

設定に誤りがある関数は定義されていないと見なして，処理を続行します。

要因

次のどれかの要因が考えられます。

- システム情報ファイル「ettrans.ini」の定義に誤りがある
- [Userfunc_DllPath] セクション又は [Userfunc_SlPath] セクションのエラーが報告された場合，出口関数を定義したライブラリに未解決シンボルがある

対策

次のどちらかの対策を実施してください。

- ログファイルに出力されたシステム情報ファイル「ettrans.ini」の不正位置に記述されている定義を確認する
- ライブラリに未解決シンボルがないか確認する

KBET0013T-W

ログファイルがオープンできないため，デフォルトのログファイルまたは標準出力に出力しました。

ログファイルを開けないため，デフォルトのログファイル又は標準出力に，ログを出力したときに出力されるメッセージです。

システムの処理

処理を続行します。

要因

次のどれかの要因が考えられます。

- 指定したログファイル名又はファイル／ディレクトリのアクセス権限が不正（READ ／ WRITE 権限が必要です）
- 開かれているファイル数が最大値を超えた

対策

ログファイル名やアクセス権限などを確認してください。

KBET0014T-S

システムでメモリ不足が発生しました。トランスレータを強制終了します。

メモリ不足で処理が続行できないときにログファイルに出力されるメッセージです。

システムの処理

処理を終了します。

対策

リソースを確保し，再度実行してください。

KBET0015T-S

システムエラーが発生しました。トランスレータを強制終了します。

システムでエラーが発生したときにログファイルに出力されるメッセージです。

システムの処理

処理を終了します。

要因

データが破壊されているおそれがあります。

対策

システム管理者に連絡してください。

KBET0016T-S

ファイルのオープンができません。

指定したファイルが開けないときにログファイルに出力されるメッセージです。メッセージに対応するファイル名がログファイルに出力されます。

システムの処理

処理を終了します。

要因

次のどれかの要因が考えられます。

- ファイル名又はファイル／ディレクトリのアクセス権限（出力側は WRITE，そのほかは READ 権限が必要です）が不正
- 別プロセス（スレッド）と重なって使用しようとした
- 開かれているファイル数が最大値を超えた

対策

ファイル名やアクセス権限などを確認してください。

KBET0017T-S

ファイルの入力処理中にエラーが発生しました。

変換する入力データの処理でエラーが発生したときにログファイルに出力されるメッセージです。メッセージに対応するファイル名がログファイルに出力されます。

システムの処理

処理を終了します。

要因

次のどちらかの要因が考えられます。

- ファイル／ディレクトリのアクセス権限（READ 権限）が不正
- 外部からファイルが破壊されている

対策

ログファイルに出力されたファイルの状態を確認し，エラー要因を修正してください。

KBET0018T-S

ファイルの出力処理中にエラーが発生しました。

変換する出力データの処理でエラーが発生したときにログファイルに出力されるメッセージです。メッセージに対応するファイル名がログファイルに出力されます。

システムの処理

処理を終了します。

要因

次のどちらかの要因が考えられます。

- ファイル／ディレクトリのアクセス権限（WRITE 権限）が不正
- 外部からファイルが破壊されている

対策

ログファイルに出力されたファイルの状態を確認し，エラー要因を修正してください。

KBET0019T-S

出力データのサイズが出力可能なメモリアドレスを超えました。

メモリ出力を指定された出力データのサイズが出力先のメモリサイズを超えたときなどにログファイルに出力されるメッセージです。メッセージに対応するメモリ開始／終了アドレスがログファイルに出力されます。

システムの処理

処理を終了します。

要因

次のどちらかの要因が考えられます。

- 出力先として確保したメモリが不足している
- 想定したサイズ以上の入力データを一括変換で出力しようとした

対策

出力先として確保するメモリサイズを増やして，再度実行してください。

KBET0020T-S

ワークファイルの作成処理でエラーが発生しました。

システムの処理

処理を終了します。

要因

次のどれかの要因が考えられます。

- トランスレータが正しくインストールされていない
- ワークディレクトリ（*Interschema のインストールディレクトリ*/tmp）のアクセス権限に READ ／ WRITE 権限がない
- OPEN されているファイル数が最大値を超えた

対策

ワークディレクトリ（*Interschema のインストールディレクトリ*/tmp）の状態を確認し，エラー要因を修正してください。

KBET0021T-E

オプションの指定に誤りがあります。

誤りのあるオプション名やフォーマット名がログファイルに出力されます。

システムの処理

変換しないで処理を終了します。

要因

コマンドの実行でオプションを指定し，処理できなかったときにログファイルに出力されるメッセージです。コマンド起動文字列に入力ミスがあります。次の要因が考えられます。

- 同じオプションを重複して定義している
- -F オプションでフォーマット名と入出力データファイル名が組で設定されていない
- -FN オプション（又は API の入出力データ差し替え情報）で入出力フォーマットが指定されていない
- API の入出力データ差し替え情報のメモリ開始／終了アドレスの大小関係が不当
- 変換対象フォーマットの入出力データが指定されていない

対策

正しいオプションを指定して，再度実行してください。

KBET0022T-W

入力フォーマットに対応するデータ以降に不当なデータがあります。

入力フォーマットとして定義されたデータ以降に，余分なデータがある場合に出力されるメッセージです。入力データのバイト位置がログファイルに出力されます。

システムの処理

処理を続行します。

要因

次のどちらかの要因が考えられます。

- 入力データが指定した MDL ファイルに対応していない
- メモリアドレスの指定に誤りがある

対策

入力データが指定した MDL（FDL）ファイルと対応しているか確認し，必要に応じて余分なデータを削除してください。

KBET0023T-E

以下のコンポーネントの作成に失敗しました。

メッセージに対応するコンポーネント名又はオフセットがログファイルに出力されます。

システムの処理

処理を終了します。

要因

コンポーネントが作成できないときにログファイルに出力されるメッセージです。
次のどちらかの要因が考えられます。
- 選択構造で選択できるコンポーネントが一つもない
- データ不正又は MDL ファイルにエラーがある（特に選択構造の選択条件が不正のおそれがある）

対策

次のどちらかの対策を実施してください。
- データが選択構造の子コンポーネントと一致するか確認する
- 必要に応じて MDL（FDL）ファイルの評価規則などを修正する

KBET0024T-E

以下のコンポーネントの出現回数が定義された値と異なります。

メッセージに対応するコンポーネント名又はオフセットがログファイルに出力されます。

システムの処理

処理を終了します。

要因

コンポーネントの出現数が，定義した回数と異なるときにログファイルに出力されるメッセージです。MDL（FDL）ファイルの定義に誤りがあります。

対策

次のどちらかの対策を実施してください。
- 入力データを確認する
- 必要に応じて，ログファイルに出力されたコンポーネントの出現回数を MDL（FDL）ファイルで修正する

KBET0025T-E

以下のコンポーネントの評価規則が真ではありません。

メッセージに対応するコンポーネント名又はオフセットがログファイルに出力されます。

システムの処理

処理を終了します。

要因

次のどちらかの要因が考えられます。
- 評価規則（条件式）の記述，又は入力データに誤りがある
- 条件を満たさないデータを変換しようとした

対策

入力データと MDL（FDL）ファイル内の評価規則（条件式）を確認し，必要に応じて評価規則（条件式）を修正してください。

KBET0026T-E

以下のコンポーネントのセパレータが見つかりません。

メッセージに対応するコンポーネント名又はオフセットがログファイルに出力されます。

システムの処理

処理を終了します。

要因

セパレータ定義に誤りがあるときにログファイルに出力されるメッセージです。入力データ又は MDL（FDL）ファイルのセパレータ定義に誤りがあります。

対策

入力データと MDL（FDL）ファイルのセパレータ定義を確認し，必要に応じてセパレータ定義を修正してください。

KBET0027T-E

以下のコンポーネントが値定義で指定された値と異なります。

メッセージに対応するコンポーネント名又はオフセットがログファイルに出力されます。

システムの処理

処理を終了します。

要因

型の値定義で「他の値が現れてもエラーにしない」を選択しなかった場合，定義していない値が出現したときにログファイルに出力されるメッセージです。次のどちらかの要因が考えられます。

- 型定義で定義していない値が出現しました。
- 値定義に誤りがあります（例えば，可変長文字データであるのにスペースの扱いがデータと値定義で異なる場合や，バイト列で記述した場合に「文字列」と「文字」との不一致が考慮されていないなど）。

対策

次のどちらかの対策を実施してください。

- 入力データを確認する
- 必要に応じて MDL（FDL）ファイルの値定義を修正するか，又は値定義を持つコンポーネントの「他の値が現れてもエラーにしない」のチェックを外す

KBET0028T-E

以下のコンポーネントのサイズ決定式の評価結果が不正です。

メッセージに対応するコンポーネント名又はオフセットがログファイルに出力されます。

システムの処理

処理を終了します。

要因

サイズ決定式の記述に誤りがあります。

対策

入力データと MDL（FDL）ファイルのサイズ決定式を確認し，必要に応じてサイズ決定式を修正してください。

KBET0029T-E

以下のコンポーネントの子コンポーネント数が最大値を超えました。

コンポーネントの子コンポーネント数が定義した数を超えたときにログファイルに出力されるメッセージです。メッセージに対応するコンポーネント名又はオフセットがログファイルに出力されます。

システムの処理

処理を終了します。

要因

FDL ファイル内の構造定義に誤りがあります。

対策

MDL（FDL）ファイルの構造定義を確認してください。

KBET0030T-E

以下のコンポーネントのマップ式の評価結果が不正です。

メッセージに対応するコンポーネント名又はオフセットがログファイルに出力されます。

システムの処理

処理を終了します。

要因

次のどれかの要因が考えられます。

- マップ式定義に誤りがある（例えば，評価順序遅延指定など）
- 入力データで必要なデータがない
- 値が想定した型と異なる

対策

入力データと MDL（FDL）ファイルのマップ式を見直し，必要に応じてマップ式を修正してください。

KBET0031T-E

要素のデータが不正です。

メッセージに対応するコンポーネント名又はオフセットがログファイルに出力されます。

システムの処理

処理を終了します。

要因

データが型定義した要素と一致していません。データ不正の内容を表 B-3 に示します。

表 B-3　データ不正の内容の説明

項目	要素の主属性	説明
データ不正	• 整数 • 実数 • 暗黙的小数部付数値	• 数字／符号／小数点／桁セパレータ／指数記号／スペース以外の文字が出現しました。 • 数字列が複数箇所にあります（スペースで区切られている）。 • 符号の出現位置・回数が型定義と一致しません。 • 桁セパレータが不正です。 桁セパレータが 3 桁ごとの区切りではありません。 桁セパレータで数値が始まっています。 整数部以外に桁セパレータがあります。 • 桁寄せ指定に一致しません。 • 指数記号が最後にあります。
	• ゾーン形式数値 • パック形式数値	• パック／ゾーン形式にデータが一致しません。 • 符号が正しくありません。
	• 日付 • 時刻 • 日付時刻	• -DTM オプション指定時に，値が年月日／時分秒ではありません（値の範囲が不正です）。
	• 混在文字列 • 1 バイト文字列 • 2 バイト文字列	• 解放文字の指定がないのに文字列中にセパレータと同じ文字があります。 • 2 バイト文字コードの要素サイズが奇数のため，埋め字（2 バイト文字）が埋まりません。
CII 標準形式でのデータ不正	－	• 規格で規定されていない値です。

（凡例）

－：該当しません。

対策

入力データを確認してください。

KBET0032T-E

要素のサイズ／桁数が最小値より小さいか，最大値を超えました。

メッセージに対応するコンポーネント名又はオフセットがログファイルに出力されます。

システムの処理

処理を終了します。

要因

出力データのサイズが型定義したサイズに達しない，又はサイズを超えました。サイズ／桁数の桁あふれ部分の内容を表 B-4 に示します。

表 B-4　サイズ／桁数の桁あふれ部分の内容の説明

項目	説明
サイズ／桁数の桁あふれ部分	• 出力要素が固定長で，データサイズが想定値を超えました。 • 小数部を持つ数値を，小数部を持たない数値に代入しようとしました。 • マップ式の演算結果が，想定する数値桁数を超えました。 • 出力側文字列中にセパレータと同じ文字があったため，解放文字を埋め込んだ分だけサイズが増加しました。 • 出力側文字列がシフトコードを使用するため，シフトコード分のサイズが増加しました。 • セパレータ区切りまでに読み込んだ要素が，最小サイズに達していません。

対策

入力データと MDL（FDL）ファイルのマップ式，出力要素の型定義を確認し，必要に応じて型定義を修正してください。

KBET0033T-E

数値型の要素の符号，小数点，指数記号が不正です。

メッセージに対応するコンポーネント名又はオフセットがログファイルに出力されます。

システムの処理

処理を終了します。

要因

入力側と出力側で文字コードが異なるため，指数記号などの文字が変換できません。

対策

入力データと MDL（FDL）ファイルの入出力要素の型定義を確認し，必要に応じて MDL（FDL）ファイルの文字コード宣言を修正してください。

KBET0034T-E

不正な文字コードがあります。

メッセージに対応するコンポーネント名又はオフセットがログファイルに出力されます。

システムの処理

処理を終了します。

要因

入力側と出力側で文字コードが異なり，かつ出力側で使用できる文字範囲が異なることが考えられます。

対策

入力データと MDL（FDL）ファイルの入出力要素の型定義を確認し，必要に応じて MDL（FDL）ファイルの文字コード宣言を修正してください。

KBET0035T-E

選択構造の評価順序が決定できません。

メッセージに対応するコンポーネント名又はオフセットがログファイルに出力されます。

システムの処理

処理を終了します。

要因

選択構造の条件設定が不正であるか，又は想定しないデータを変換しようとしました。

対策

入力データを確認し，必要に応じて MDL（FDL）ファイルの評価順序決定式を修正してください。

KBET0036T-E

トランスレータ指示指定に誤りがあります。

メッセージに対応するコンポーネント名又はオフセットがログファイルに出力されます。

システムの処理

処理を終了します。

要因

セパレータの指定で文字列／バイト列以外の値を代入しようとしました。

対策

入力データを確認し，必要に応じて変換情報を定義したときのトランスレータ指示指定を修正してください。

KBET0037T-E

UN/EDIFACT **のヘッダ情報が不正です。**

メッセージに対応するヘッダ名がログファイルに出力されます。

システムの処理

処理を終了します。

要因

データ不正，INTERACTIVE メッセージの運用に誤りがあります。

対策

ヘッダの内容が正しく設定されているか確認してください。

KBET0038T-E

シンタックス ID **が不正です。**

システムの処理

処理を終了します。

要因

トランスレータが対応しない CII シンタックスルールのバージョンのデータを変換しようとしました。

対策

CII のバージョン 1.51，2.10，又は 3.00 形式のデータで再度変換してください。Interschema の CII シンタックスルールのサポート範囲については，「付録 G　CII シンタックスルールのサポート範囲」を参照してください。

KBET0039T-E

CII データが非透過モードのため変換できません。

システムの処理

処理を終了します。

要因

非透過モードデータを変換しようとしました。

対策

CII の透過モード形式のデータで再度変換してください。

KBET0040T-E

CII データの分割区分が不正です。

システムの処理

処理を終了します。

要因

次のどちらかの要因が考えられます。

- 分割区分がシーケンシャルになっていない
- 最終レコードの分割区分が 9（メッセージ）又は I（バイナリデータ）ではない

対策

CII 標準形式のデータが分割モードで正しく作成されているか確認してください。

KBET0041T-E

CII メッセージまたはバイナリデータのシーケンス番号が不正です。

システムの処理

処理を終了します。

要因

次のどれかの要因が考えられます。

- シーケンス番号が昇順ではない（メッセージ／バイナリデータ混在のときに別々に数えられているなど）
- メッセージグループ・トレーラのシーケンス番号とメッセージ又はバイナリデータ数が一致していない
- 0 件メッセージのメッセージグループ・トレーラのシーケンス番号が 0 以外になっている

対策

メッセージ又はバイナリデータのシーケンス番号が昇順であるか，又はメッセージ

やバイナリデータ数がメッセージグループ・トレーラに設定されているか確認してください。

KBET0042T-E

CII **データにメッセージグループ・ヘッダがありません。**

システムの処理

処理を終了します。

要因

CII 標準形式のデータが入力データの場合は，CII 標準形式以外のデータが入力されたか，又はデータが破壊されています。出力データの場合は，辞書の不当な修正がありました。

対策

CII 標準形式のデータが入力データの場合はデータを確認してください。出力データの場合は MDL（FDL）ファイル構造とマップ式を確認してください。

KBET0043T-E

CII **データにメッセージグループ・トレーラがありません。**

システムの処理

処理を終了します。

要因

CII 標準形式のデータが入力データの場合は，次のどちらかの要因が考えられます。

- データ構造に誤りがある
- データが破壊されている

CII 標準形式のデータが出力データの場合は，辞書の不当な修正があったことが考えられます。

対策

CII 標準形式のデータが入力データの場合はデータを確認してください。出力データの場合は MDL（FDL）ファイル構造とマップ式を確認してください。

KBET0044T-E

CII **データにメッセージヘッダがありません。**

システムの処理

処理を終了します。

要因

CII 標準形式のデータが入力の場合はレコード区分に誤りがあるか，又はデータが破壊されています。出力データの場合は辞書の不当な修正がありました。

対策

CII 標準形式のデータが入力データの場合はデータを確認してください。出力データ

の場合は MDL（FDL）ファイル構造とマップ式を確認してください。

KBET0045T-E

CII データにバイナリデータ・トレーラがありません。

システムの処理

処理を終了します。

要因

CII 標準形式のデータが入力データの場合はデータが破壊されています。出力データの場合は辞書の不当な修正がありました。

対策

CII 標準形式のデータが入力の場合はデータを確認します。出力の場合は MDL（FDL）ファイル構造とマップ式を確認してください。

KBET0046T-E

縮小モードの CII データにマルチ明細のネストがあります。

システムの処理

処理を終了します。

要因

データ中に縮小モードでのマルチ明細ネストがあります。

対策

データを拡張モード形式とするか，又はマルチ明細のネストを解除して再度変換してください。

KBET0047T-E

CII データに未定義のデータタグがあります。

システムの処理

処理を終了します。

要因

タグ番号が決められた範囲外か，又は対応していない指示子が使用されています。

対策

次のどちらかの対策を実施してください。

- CII 標準形式のデータのタグ番号を確認する
- トランスレータで対応しないタグを取り除く

KBET0048T-E

CII データの構造が不正です。

システムの処理

処理を終了します。

要因

0 件メッセージでメッセージグループ・ヘッダやトレーラ以外のデータがあるか，又は受信確認，エラー情報メッセージのデータがありません。

対策

運用メッセージデータの内容を確認してください。

KBET0049T-E

CII **データのチェックサム項目の値が数値ではありません。**

メッセージに対応するタグ番号又はコンポーネント名とオフセットがログファイルに出力されます。

システムの処理

処理を終了します。

要因

チェックサム項目の設定に誤りがあるか，又は項目のデータが不正です。

対策

ログファイルに出力されたコンポーネントの内容，チェックサム項目のタグ番号を確認してください。

KBET0050T-E

CII **データの項目長が不正です。**

メッセージに対応するコンポーネント名又はオフセットがログファイルに出力されます。

システムの処理

処理を終了します。

要因

データ項目長が CII で規定されたサイズを超えているか，又は 0 です。

対策

ログファイルに出力されたコンポーネントのデータサイズを確認してください。

KBET0051T-E

CII **データの同報ヘッダが不正です。**

システムの処理

処理を終了します。

要因

同報ヘッダの継続区分，個数の設定に誤りがあります。

対策

同報ヘッダの継続区分，同報個数を確認してください。

KBET0052T-E

CII データの変換処理でエラーが発生しました。

システムの処理

処理を終了します。

要因

変換正常処理時の受信確認，エラー情報メッセージの出力でエラーが発生したか，又はメッセージ中に出力できる TFD 項目がありません。

対策

このメッセージ以外に出力された内容（要素データ不正など），又はマップ式及び入力データを確認してください。

KBET0053T-E

トランスレータと MDL のバージョンが不整合です。

システムの処理

処理を終了します。

要因

MDL ファイルのバージョンがトランスレータで扱えるバージョンより新しくなっています

対策

トランスレータと MDL ファイルのバージョンを合わせてください。

KBET0054T-E

未検証の MDL が指定されました。

システムの処理

処理を終了します。

要因

MDL ファイルの検証を実行していません。又は MDL ファイル修正後に MDL ファイルの検証を実行していません。

対策

MDL エディタで MDL ファイルのチェックコマンドを実行してから，再度実行してください。

KBET0055T-W

出口関数でエラーが発生しました。

出口関数名と出口関数の戻り値がログファイルに出力されます。

システムの処理

処理を続行します。

要因

出口関数の処理でエラーが発生しました。

対策

出口関数の処理，引数としたコンポーネントの値などを確認してください。

KBET0056T-E

出口関数でエラーが発生しました。

出口関数名と出口関数の戻り値がログファイルに出力されます。

システムの処理

処理を終了します。

要因

出口関数の処理でエラーが発生しました。

対策

出口関数の処理，引数としたコンポーネントの値などを確認してください。

KBET0057T-E

グループ単位出力指定されているコンポーネントに対応するコンポーネントがありません。

グループ単位出力指定されているコンポーネント名がログファイルに出力されます。

システムの処理

処理を終了します。

要因

選択構造の子コンポーネント間で対応関係を設定してある場合に，対応するコンポーネントが選択されませんでした。グループ単位出力指定に誤りがあるか，又は選択構造の子コンポーネントを選択する条件が正しくありません。

対策

グループ単位出力指定が入出力フォーマット間で正しく指定されているか確認してください。

KBET0058T-E

Sea-NACCS **データの電文長が許容範囲を超えました。**

業務コードと電文長がログファイルに出力されます。

システムの処理

処理を終了します。

要因

規定外のメッセージ構造であるため，電文長が上限を超えました。

対策

Sea-NACCS の処理要求電文で対応する業務メッセージ構造を確認してください。

KBET0059T-E

RDB 接続フォーマットは変換できません。

システムの処理

処理を終了します。

要因

RDB 接続フォーマットを含む MDL ファイルで変換しようとしました。

対策

RDB 接続フォーマットを含む MDL は，電子データ交換ツール 01-04 で変換してください。

KBET0060T-E

XML パーサによる解析でエラーが発生しました。

XML パーサが出力するエラーコードとメッセージがログファイルに出力されます。

システムの処理

処理を終了します。

要因

入力 XML 文書に誤りがあります。

対策

入力 XML 文書を確認してください。

KBET0061T-E

XML ドキュメント出力時にエラーが発生しました。

システムの処理

処理を終了します。

要因

XML パーサのドキュメント出力機能実行時にエラーが発生しました。

次のどちらかの要因が考えられます。

- ファイル／ディレクトリのアクセス権限（WRITE 権限）が不正
- オープンできるファイル数の制限を超えた
- 外部からファイルが破壊されている

対策

出力データファイルの状態を確認し，エラー要因を修正してください。

KBET0062T-E

不正なコンポーネントデータを検出しました。

メッセージに対応するコンポーネント名がログファイルに出力されます。

システムの処理

処理を終了します。

要因

入力 XML 文書が MDL ファイルの構造定義と一致しません。

対策

入力 XML 文書を確認してください。

KBET0063T-E

XML 属性の評価結果の値が不正です。

メッセージに対応する属性のコンポーネント名がログファイルに出力されます。

システムの処理

処理を終了します。

要因

変換後の XML 属性データが，MDL ファイル中の固定値（#FIXED 属性の場合），又は列挙値（属性値の候補が列挙型の場合）と一致しない場合にログファイルに出力されます。

次のどちらかの要因が考えられます。

- 固定値，又は列挙値で定義していない値が出現した
- MDL ファイル作成時に読み込んだ DTD に誤りがある

対策

次のどちらかの対策を実施してください。

- 入力データを確認
- 必要に応じて MDL ファイルに読み込む DTD を修正するか，エラーになった XML 属性のマップ式を削除

KBET0064T-E

XML 処理命令のターゲット名に不正な文字コードがあります。

システムの処理

処理を終了します。

要因

MDL ファイル中に定義された XML 処理命令のターゲット名に，XML 1.0 で許されない不正な文字列が指定されています。

対策

MDL ファイルに定義した処理命令を確認してください。

KBET0065T-E

以下のコンポーネントの出現回数決定式の評価結果が不正です。

メッセージに対応するコンポーネント名とオフセットがログファイルに出力されます。

システムの処理

処理を終了します。

要因

出現回数決定式の記述に誤りがあります。

対策

入力データと MDL ファイルの出現回数決定式を確認し，必要に応じて出現回数決定式を修正してください。

KBET0066T-E

MDL の内容が不正です。

システムの処理

処理を終了します。

要因

MDL ファイルをワークステーションの OS のマシンへ転送する手順に誤りがあったことが考えられます。

対策

MDL を確認してください。ワークステーションの OS のマシンへ転送する場合は，バイナリデータとして転送してください。

KBET0067T-E

変数式の評価結果が不正です。

メッセージに対応するコンポーネント名又はオフセットがログファイルに出力されます。

システムの処理

処理を終了します。

要因

次のどれかの要因が考えられます。

- 変数式定義に誤りがある
- 入力データで必要なデータがない
- 値が想定した型と異なる

対策

入力データと MDL（FDL）ファイルの変数式を見直し，必要に応じて変数式を修正してください。

KBET0068T-S

変換できない MDL が指定されました。

システムの処理

処理を終了します。

要因

変換に必要なライセンス製品がインストールされていません。

対策

必要なライセンス製品をインストール後，再度変換してください。MDL に含まれるフォーマットの規格種別と必要ライセンス製品の対応を次に示します。

表 B-5　規格種別と必要ライセンス製品

規格種別	必要なライセンス製品
CII CII3	uCosminexus Interschema CII ライセンス
EDIFACT EDIFACT4	uCosminexus Interschema EDIFACT ライセンス
Sea-NACCS Sea-NACCS 用 EDIFACT	uCosminexus Interschema 海上貨物通関ライセンス
XML	uCosminexus Interschema XML ライセンス

KBET0069T-W

出口関数が二重に定義されました。

メッセージに対応する組み込み関数の ID が，ログファイルに出力されます。

システムの処理

設定に誤りがある関数は，定義されていないと見なして，処理を続行します。

要因

同じ組み込み関数の ID に対する出口関数が，[Userfunc_JavaMethodName] セクション及び [Userfunc_ExitfuncName] セクションの両方に定義されています。

対策

報告された出口関数を使用する場合は詳細情報を確認して，システム情報ファイルを修正後に，再度実行してください。

KBET0070T-E

Java 出口関数が見つかりません。

メッセージに対応する組み込み関数名が，ログファイルに出力されます。

システムの処理

処理を終了します。

要因

使用している Java 言語の出口関数が，Java 実行環境のクラスパスに見つかりません。

対策

詳細情報を確認して，対応する Java 言語の出口関数を Java 実行環境のクラスパス内に配置後に，再度実行してください。

KBET0071T-E

出口関数で未知の例外が発生しました。

メッセージに対応する組み込み関数名が，ログファイルに出力されます。

システムの処理

処理を終了します。

要因

Java 言語の出口関数から，ExitFuncException クラス又は ExitFuncWarning クラス以外の例外が送出されました。

対策

ExitFuncException クラス又は ExitFuncWarning クラスどちらかの例外をスローするように出口関数を作成し直して，再度実行してください。

KBET0072T-E

出口関数の戻り値が不正です。

メッセージに対応する組み込み関数名が，ログファイルに出力されます。

システムの処理

処理を終了します。

要因

許可されていない値が，出口関数又は ExitFuncWarning クラス（Java 言語の出口関数の場合だけ）の戻り値として指定されました。

対策

正しい戻り値を返すように出口関数を作成し直して，再度実行してください。

KBET0073T-S

ＭＤＬ 情報またはスレッド固有情報が不正です。

システムの処理

処理を終了します。

要因

MDL 情報又はスレッド固有情報が，別のスレッドで使用中です。スレッド間の排他漏れ，又はユーザプログラムの構築が不正です。

対策

API の使用方法は，Interschema が示す API の実行順序に従っているか，マルチスレッド実行時に必要な情報の排他処理が行われているかを確認してください。

KBET0078T-E

Sea-NACCS データの電文長が不正です。

該当するデータのオフセットが，ログファイルに出力されます。

システムの処理

処理を終了します。

要因

Sea-NACCS EDI 電文の電文長がないか，又は数値ではありません。

対策

Sea-NACCS データの電文長を確認してください。

付録 B.4　データ変換処理 API（Java 言語）の例外

エラー発生時に通知されるデータ変換処理 API（Java 言語）の例外について説明します。

KBET0001J-E

パラメータの指定に誤りがあります。詳細：aa=bb

aa：パラメータ名

bb：パラメータ値

要因

指定したパラメータが不正です。

対処

詳細情報を確認して，エラーの原因を取り除いた後，再度実行してください。

KBET0002J-E

文字列の変換に失敗しました。

要因

文字列の変換に失敗しました。

対処

エラーの原因を取り除いた後，再度実行してください。

KBET0003J-S

指定されたファイルが存在しないか，アクセス権がありません。

要因

指定のファイルが存在しないか，又はアクセス権がありません。

対処

指定のファイルを確認して，再度実行してください。

KBET0004J-S

Java VM **のバージョンが不正です。**

要因

Java Virtual Machine（Java VM）のバージョンが不正です。

対処

Java Virtual Machine（Java VM）のバージョンを確認してください。

KBET0005J-S

システムでメモリ不足が発生しました。

要因

メモリが不足しました。

対処

メモリ容量を確保した後，再度実行してください。

KBET0006J-S

システムエラーが発生しました。

要因

処理中にシステムエラーが発生しました。

対処

システム管理者に連絡してください。

KBET0007J-S

予期せぬエラーが発生しました。

要因

予期しないエラーが発生しました。

対処

システム管理者に連絡してください。

KBET0008J-S

予期せぬエラーが発生しました。詳細：aa

aa：原因となった例外の詳細メッセージ

要因

予期しないエラーが発生しました。

対処

システム管理者に連絡してください。

KBET0009J-E

ライブラリの初期化に失敗しました。詳細：aa

aa：原因となった例外の詳細メッセージ

要因

ライブラリの初期化処理でエラーが発生しました。

対処

詳細情報を確認して，エラーの原因を取り除いた後，再度実行してください。

KBET0010J-S

メッセージリソースが見つかりません。

要因

エラーメッセージ用のリソースファイルが見つかりません。

対処

システム管理者に連絡してください。

KBET0011J-E

オプションの指定に誤りがあります。

要因

指定した実行時オプションが不正です。

対処

ログを確認して，エラーの原因を取り除いた後，再度実行してください。

KBET0012J-E

変換処理中にエラーが発生しました。

要因

変換処理中にエラーが発生しました。

対処

拡張エラー情報を確認して，エラーの原因を取り除いた後，再度実行してください。

KBET0013J-E

変換処理中にエラーが発生しました。詳細：aa

aa：エラーの原因を示す詳細メッセージ

要因

変換処理中にエラーが発生しました。

対処

詳細情報を確認して，エラーの原因を取り除いた後，再度実行してください。

KBET0014J-S

出口関数でエラーが発生しました。

要因

出口関数でエラーが発生しました。

対処

ログを確認して，エラーの原因を取り除いた後，再度実行してください。

KBET0015J-S

トランスレータの生成に失敗しました。

要因

トランスレータオブジェクトの生成に失敗しました。

対処

システム管理者に連絡してください。

KBET0016J-S

トランスレータの更新に失敗しました。

要因

トランスレータオブジェクトの更新に失敗しました。

対処

システム管理者に連絡してください。

KBET0017J-S

トランスレータの解放に失敗しました。

要因

トランスレータオブジェクトの解放に失敗しました。

対処

システム管理者に連絡してください。

KBET0018J-E

MDL **ファイルが不正です。**

要因

MDL ファイルが未検証，バージョン不正，又は不正です。

対処

指定した MDL ファイルを確認して，再度実行してください。

KBET0019J-S

MDL **情報の生成に失敗しました。**

要因

MDL 情報の生成に失敗しました。

対処

システム管理者に連絡してください。

KBET0020J-S

MDL **情報の解放に失敗しました。**

要因

MDL ファイルの解放に失敗しました。

対処

システム管理者に連絡してください。

KBET0021J-E

FDL または MDL ファイルが不正です。

要因

FDL ファイル又は MDL ファイルの内容が不正です。

対処

指定したファイルを確認して，再度実行してください。

KBET0022J-E

コード変換で不正な文字コードを検出しました。

要因

DL プロパティ情報の生成時に不正な文字コードを検出しました。

対処

DL プロパティ情報のユーザコメント，及びユーザバージョン情報の文字コードを確認して，再度実行してください。

KBET0023J-S

DL プロパティ情報の生成に失敗しました。

要因

DL プロパティ情報の生成に失敗しました。

対処

システム管理者に連絡してください。

KBET0024J-E

データオブジェクトからバイト配列への変換に失敗しました。詳細：aa

aa：原因となった例外の詳細メッセージ

要因

データオブジェクトをバイト配列に変換できませんでした。

対処

詳細情報を確認して，エラーの原因を取り除いた後，再度実行してください。

KBET0025J-E

バイト配列からデータオブジェクトへの変換に失敗しました。詳細：aa

aa：原因となった例外の詳細メッセージ

要因

バイト配列をデータオブジェクトに変換できませんでした。

対処

詳細情報を確認して，エラーの原因を取り除いた後，再度実行してください。

KBET0026J-S

変換できない MDL が指定されました。

要因

変換に必要なライセンス製品がインストールされていません。

対処

必要なライセンス製品をインストール後，再度実行してください。MDL に含まれるフォーマットの規格種別と必要ライセンス製品の対応については，「付録 B.3　メッセージ」の「KBET0068T-S」を参照してください。

付録 C　トランスレータの機能差異と電子データ交換ツールからの移行上の注意

Windows 版のトランスレータとワークステーションの OS のトランスレータとの機能差異と，電子データ交換ツールから Interschema バージョン 1 へ移行する場合の注意について説明します。

なお，Interschema バージョン 1 から Interschema バージョン 2 以降への API（C 言語）の移行には，Interschema バージョン 1 との互換 API が使用できます。互換 API については，「付録 D　Interschema バージョン 1 との互換 API」を参照してください。

付録 C.1　トランスレータの機能差異

Windows 版とワークステーションの OS のトランスレータとの機能差異を次に示します。Windows 版とワークステーションの OS との機能差異に注意して，トランスレータを使用してください。

表 C-1　トランスレータの機能差異

項番	内容	Windows 版	ワークステーションの OS
1	インストールパス	レジストリによる設定に従います。	環境変数 INTERSCHEMA で設定します。未設定時は「/opt/hitachi/interschema」が想定されます。
2	ファイル名の指定	• 空白を含むファイル，又はフォルダ名をパラメタとして指定する場合，項目全体を「"」で囲みます（パラメタ指定は除く）。	• 同左。また，ファイル名に「"」又は「¥」を含む場合は，「¥」をエスケープ文字とします。 • ファイル名は，256 バイト以内に制限されます。
3	ユーザ組み込み関数の定義方法	• ユーザが組み込み関数機能を実装した DLL を作成し，トランスレータ実行時に動的にバインドします。 • DLL パスは，システム情報ファイル ettrans.ini の [Usefunc_DllPath] セクションに記述します。	• 共用ライブラリのバインドです。 • パスは [Usefunc_SlPath] セクションに記述します。 • システム情報ファイルサイズの上限はありません。
4	ログファイル	変換日時，スレッド ID，戻り値，エラーメッセージ，及び付加情報を出力します。	左記の内容に加えてプロセス ID が出力されます。

付録 C.2　電子データ交換ツールからの移行上の注意

Interschema と電子データ交換ツールとの機能差異，及び電子データ交換ツールからの移行上の注意について説明します。

(1) 電子データ交換ツールとの機能差異

Interschema と電子データ交換ツールには，次のような機能差異があります。なお，機能差異は電子データ交換ツールが持つ機能について，Interschema と比較しています。

表 C-2　電子データ交換ツールとの機能差異

項番	分類	電子データ交換ツール	Interschema
1	トランスレータ	次の機能をサポートしています。 • CII 対応トランスレータ互換機能（/EDIFT） • ISO8859 互換オプション	左記の機能をサポートしていません。 ※2
2	エディタ全般	チェック時の異種属性代入チェックオプションがあります。	左記の機能をサポートしていません(常に異種属性間の代入式を許す)。
3	コンポーネント名	最大 64 バイト。英数字，1 バイト仮名文字，及び 2 バイト文字から構成されます。先頭が数字又は数字だけの名称は使用できません。	フォーマット名は，256 バイト以内です。そのほかは，4,096 バイトまでです。 使用できる文字は，XML に従います。ただし，2 バイト文字の制限はありませんが，範囲はシフト JIS に限ります。
4	API	• ETtransExec • ETtransExecS • ETtransErrorInfo	左記 API を ETtransExec に統合しました。※3
5	日付時刻型※1	i_num 型，及び 10 進整数として処理します。	中間型は i_dtm，演算時に 10 進数化して処理します。 パート値の定義は数値型と同じだが，型全体としては文字列型と同じに扱われます。
6	ユーザ組み込み関数名	64 バイト以内。半角英数字，先頭英字で指定します。	システム関数との重複を避けるため，先頭文字は下記のとおりとします。 U，u，W，w，X，x，Y，y，Z，z
7	サイズ決定式	入力フォーマットだけ有効となります。	入力／出力フォーマットの両方で有効となります。
8	トランスレータ及び戻り値	• コマンド及び API：(0x)0nxx • テスト用データ：0x0108(ワーニング)	• コマンド及び API：(0x)0n0000xx • テスト用データ：0x00000008（インフォメーション）
9	実行時オプション	/xxx（Windows 版だけ）	-xxx
10	ログファイルの初期値	「*電子データ交換ツールのインストールディレクトリ*¥bin」下へ出力（Windows 版だけ）	「*Interschema のインストールディレクトリ*/log」下へ出力

項番	分類	電子データ交換ツール	Interschema
11	不要文字削除指定	入力フォーマットでは無視され，入力文字列の埋め字を除いてデータと見なされます。	入力フォーマットの文字列に対して不要文字削除指定がある場合は，埋め字を除いてデータと見なし，不要文字削除指定がない場合は埋め字を含めてデータとして扱います。 電子データ交換ツールで作成されたMDL ファイルの読み込み時，入力フォーマットは不要文字削除指定ありと見なします。
12	Unicode文字列のバイトオーダマーク	型（文字列）ごとに付けて出力します。	型（文字列）のデータとして付けないで出力します。ただし，出力がXML 文書の場合は，文書の先頭にバイトオーダマークが出力されます。 エンディアンはフォーマットの指定に従います。

注※ 1

電子データ交換ツールの日付型と時刻型は，数値属性です。Interschema では，数値型とは別に日付時刻型をサポートしています。

電子データ交換ツールの日付型と時刻型は，数値属性のため，埋め字は 0 又はスペース，1 バイト文字コードで表しましたが，Interschema の日付型，時刻型，及び日付時刻型は，文字列型の属性を持ちます。埋め字は任意で，混在文字コードが使用できます。

日付型と時刻型を使用した電子データ交換ツールの MDL に従って，Interschema で変換を実行した場合，電子データ交換ツールの日付型と時刻型は，自動的に Interschema の日付時刻型に置き換わるため，変換した結果が異なることがあります。サイズ，左右寄せなどの要素の属性を確認した上で，変換してください。

注※ 2

Interschema は，CII 対応トランスレータの互換機能をサポートしていません。CII 対応トランスレータの定義情報による変換は，電子データ交換ツールで実行してください。

注※ 3

注意事項の詳細は，「付録 C.4　データ変換処理 API を使用したプログラムの移行上の注意」を参照してください。

付録 C.3　出口関数移行上の注意点

ユーザ組み込み関数に対する出口関数は，インクルードファイル「ETexit.h」の ETEXITDATA 構造体の定義が変更になっているので，必ず再コンパイルしてください。

また，C コンパイラではなく，C++ コンパイラでコンパイルする場合は，関数名がシステム情報ファイル「ettrans.ini」の［Userfunc_ExitfuncName］セクションで定義した

名前と同じになるように，「extern “ C ”」を指定して作成してください。

システム情報ファイル「ettrans.ini」での出口関数の定義は，そのまま使用できます。

付録 C.4　データ変換処理 API を使用したプログラムの移行上の注意

電子データ交換ツールが提供している ETtransExecS 関数と ETtransErrorInfo 関数は，Interschema バージョン 1 では ETtransExec 関数へ統合されています。ETtransExec 関数は，引数も変更になっています。また，ETtransExec 関数，及び ETtransExecT 関数のパラメタとして使用する ETTRANSADRSLIST 構造体が変更になっているため，プログラムの変更が必要です。

データ変換処理関数 API を含む共用ライブラリは C++ で作成しているので，データ変換処理関数 API を使用したプログラムも C++ コンパイラでのコンパイル・リンクが必要です。ワークステーションの OS の場合は，makefile のサンプルファイルを参考に makefile を作成してください。makefile のサンプルファイルの格納場所は，リリースノートを参照してください。HP-UX の場合，実行時に環境変数 SHLIB_PATH で指定したディレクトリから共用ライブラリを検索するために，リンカへのオプションとして必ず「+s」を指定してください。

(a) ETtransExec 関数引数

第 6 引数「int iFNFlag」，第 7 引数「ETTRANSERRLIST *pErrList」が追加になっています。

第 6 引数 iFNFlag には，次の値を指定してください。

- ETtransExec 関数で処理していたもの：ET_OP_FNOFF
- ETtransExecS 関数で処理していたもの：ET_OP_FNON

第 7 引数 pErrList には，次の値を指定してください。

- ETtransErrorInfo 関数でエラー情報を取得していた場合：ETtransErrorInfo 関数の第 2 引数と同じもの
- ETtransErrorInfo 関数未使用だった場合：NULL

上記に合わせて，ETtransExecS 関数は ETtransExec 関数に置き換えて，ETtransErrorInfo 関数の実行処理は削除してください。

(b) ETTRANSADRSLIST 構造体の変更

ファイル指定の場合のファイル名を表すメンバ（電子データ交換ツールでの DataFile）と，メモリ指定の開始アドレスを表すメンバ（電子データ交換ツールでの StartAddress）が統合されて，一つのメンバ（Interschema バージョン 1 での DataAddress）になっています。次に示すように，構造体のメンバ名を変更してくださ

い。

- ファイル指定の場合
  DataFile を DataAddress に変更し，StartAddress メンバ操作部分を削除する
- メモリ指定の場合
  StartAddress を DataAddress に変更し，DataFile メンバ操作部分を削除する

なお，Interschema バージョン 1 ではこのほかにもメンバの追加がありますが，電子データ交換ツールの機能範囲内で変換する場合は，設定しなくてもかまいません。詳細については，「付録 D.3　ETtransExec（トランスレータの起動）」を参照してください。

付録 D　Interschema バージョン 1 との互換 API

Interschema バージョン 1 との互換 API について説明します。

互換 API の使用は，Interschema バージョン 1 から使用しているユーザが対象です。新しく API を作成する場合には，データ変換処理 API を使用します。データ変換処理 API の概要については，「8.　インタフェース」を参照してください。データ変換処理 API の詳細については，「9.　データ変換処理 API（C 言語）」又は「10.　データ変換処理 API（Java 言語）」を参照してください。

Interschema バージョン 1 との互換 API は，Interschema バージョン 1 で使用しているデータ変換処理 API です。

この互換 API と 8 章，9 章，及び 10 章のデータ変換処理 API は，混在させて使用することはできません。

次に互換 API の関数一覧を示します。

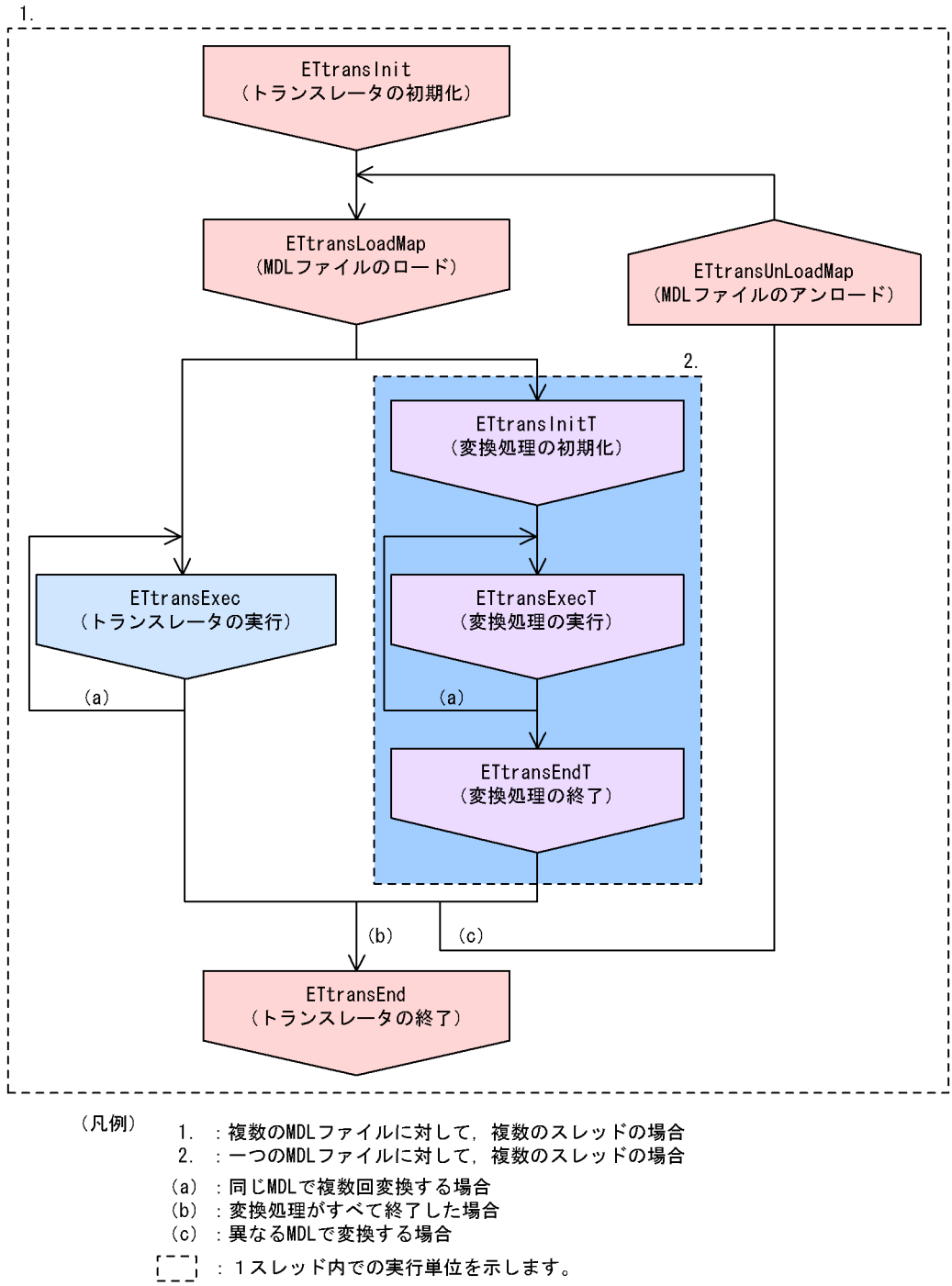
表 D-1　互換 API の関数一覧

関数名	機能概要
ETtransInit	トランスレータの初期化
ETtransLoadMap	MDL ファイルのロード
ETtransExec	トランスレータの起動
ETtransUnLoadMap	MDL ファイルのアンロード
ETtransEnd	トランスレータの終了
ETtransInitT	変換処理の初期化※
ETtransExecT	変換処理の実行※
ETtransEndT	変換処理の終了※

注※
　変換処理の実行単位でスレッド化する場合に使用します。

互換 API の関数の実行順序を，次に示します。

図 D-1　互換 API の関数の実行順序



（凡例）

1.　：複数のMDLファイルに対して，複数のスレッドの場合
2.　：一つのMDLファイルに対して，複数のスレッドの場合
(a)　：同じMDLで複数回変換する場合
(b)　：変換処理がすべて終了した場合
(c)　：異なるMDLで変換する場合
：１スレッド内での実行単位を示します。

互換 API の関数は，マルチスレッドでのデータ変換に使用できます。ただし，次の点に注意してください。

- 1. の場合，各スレッドでは，必ず ETransInit 関数と ETtransEnd 関数を実行してください。
- 2. の場合，変換処理の実行部分をマルチスレッド化する場合は，ETtransInitT 関数，ETtransExecT 関数，及び ETtransEndT 関数を使用してください。

互換 API の関数の仕様については，次の順序で説明します。

形式
関数の記述形式を示します。

引数
関数の引数及び入出力の種別を示します。

説明
関数の機能について説明します。

戻り値
関数の戻り値を示します。

付録 D.1　ETransInit（トランスレータの初期化）

形式

```
#include  "ETtrans.h"

int ETtransInit (char  **c, char  *pszLogFile)
```

引数

引数	種別	内容
c	出力	システムの管理情報のアドレスが格納されます。
pszLogFile	入力	ログファイル名を指定します。

説明

トランスレータを初期化します。
ETtransExec 関数又は ETtransExecT 関数で変換を実行する前にこの関数を実行し，この関数が正常に終了したことを確認する必要があります。
引数 c で取得したシステムの管理情報のアドレスは，ほかのデータ変換処理の関数へ渡す必要があります。
この関数を呼び出したら，ETtransEnd 関数が呼び出されるまで，すべての関数処理のログ情報は引数 pszLogFile で指定したファイルに出力されます。ただし，ETtransExec 関数と ETtransExecT 関数では，ログ情報の出力先を別ファイルに指定できます。詳細については，ETtransExec 関数と ETtransExecT 関数の説明を参照してください。
引数 pszLogFile に指定したファイルを作成又は更新できなかった場合や引数 pszLogFile に NULL を指定した場合，エラー情報はデフォルトのログファイル

「errlog.txt」に出力されます。デフォルトのログファイルを作成又は更新できなかった場合は，標準出力に出力されます。
この関数内でログファイルに情報を出力する場合，ログファイルの最大サイズは初期値（1MB）となります。ただし，「ettrans.ini」ファイルでログ出力抑止を指定している場合は，引数 pszLogFile の指定に関係なく，ログは出力されません。

戻り値

戻り値	内容
0x00000000	正常に終了しました。
0x01000010※	指定されたログファイルをオープンできないため，ログを初期値のログファイル名で出力したか又は標準出力に出力しました。
0x01000040※	出口関数の定義に不正があります。
0x01000100※	「ettrans.ini」の定義に不正があります。
0x04000000	トランスレータを初期化できませんでした。
0x04000002	メモリ不足です。

注※

0x01000010，0x01000040，及び 0x01000100 は，ほかの値と同時に設定される場合があります。

付録 D.2　ETtransLoadMap（MDL ファイルのロード）

形式

```
#include  "ETtrans.h"

int  ETtransLoadMap (char  *c, char  *pszMDLFile)
```

引数

引数	種別	内容
c	入力	システムの管理情報のアドレス（ETtransInit 関数の引数 c で取得したアドレス）を指定します。
pszMDLFile	入力	ロードしたい MDL ファイル名を指定します。

説明

MDL ファイルをメモリ上にロードします。
ETtransExec 関数又は ETtransExecT 関数を実行する前に，この関数で MDL ファイルをロードしておく必要があります。
既に MDL ファイルがロードされている状態でこの関数を実行するとエラーとなります。

戻り値

戻り値	内容
0x00000000	正常に終了しました。
0x02000000	MDL ファイルの定義内容が検証されていない，MDL ファイルのバージョン不整合，又は MDL ファイルの内容が不正です。
0x04000000	MDL ファイルをロードできませんでした。
0x04000001	指定された MDL ファイルが見付からない，又はファイルへのアクセス権がありません。
0x04000002	メモリ不足です。
0x04000010	変換できない MDL です（対応するライセンスがありません）。

付録 D.3　ETtransExec（トランスレータの起動）

形式

```
#include  "ETtrans.h"

int  ETtransExec(char  *c, ETTRANSADRSLIST  *pAdrsList,
                 char  *pszLogFile, char  *pszOptList,
                 int  iFileAppendFlag,  int  iFNFlag,
                 ETTRANSERRLIST  *pErrList)
```

引数

引数	種別	内容
c	入力	システムの管理情報が格納されているアドレス（ETtransInit 関数の引数 c で取得したアドレス）を指定します。
pAdrsList	入力	入出力データ差し替え情報の配列を指定します。
pszLogFile	入力	ログファイル名を指定します。
pszOptList	入力	コマンドラインで指定できる実行時オプションを指定します。オプションの詳細については，「5.4.1(3)　引数」を参照してください。
iFileAppendFlag	入力	ファイルの出力方法を指定します。 • ET_FA_UPDATE：新規又は上書き • ET_FA_APPEND：追加書き
iFNFlag	入力	MDL 内のフォーマットのうち，変換対象とするフォーマットを特定するかどうかを指定します。 • ET_OP_FNOFF：指定なし • ET_OP_FNON：指定あり
pErrList	出力	エラー情報リストの出力先を指定します。

説明

引数 pszOptList に指定したオプションでトランスレータを起動し，ETtransLoadMap 関数でロードした MDL ファイルに定義されている内容に従ってデータを変換します。

MDL ファイルがロードされていない状態でこの関数を実行するとエラーとなります。
引数 pAdrsList は，MDL ファイル内に定義した入出力ファイルを別ファイル又はメモリデータに差し替える場合に指定します。この引数でデータを差し替えるフォーマット名と，対応するデータのファイル名，又はメモリアドレスを指定します。
データを差し替えるフォーマット名は複数指定できます。また，差し替え対象となるフォーマットの終了を示すために，ETTRANSADRSLIST 配列の最後の要素の FormatName メンバを NULL にする必要があります。
データの差し替えを指定されたフォーマットに対しては，MDL ファイル内で定義された入出力データファイル名は無視されます。入力データと出力データのファイルは，同じファイル名を指定するとエラーとなります。また，MDL ファイル内で定義された入出力データファイル名と同じファイル名を指定した場合もエラーとなるので注意してください。
構造体 ETTRANSADRSLIST は，ヘッダファイル ETtrans.h 内で定義されています。ETtrans.h の内容を次に示します。

```
typedef struct {
  char *FormatName;  /* 差し替え対象となるフォーマット名 */
  int  DataKind;     /* データ種別（ET_DK_*）*/
  char *DataAddress; /* 入出力データファイル名（ファイル指定）*/
                     /* 入出力データの開始アドレス（メモリ指定）*/
  char *EndAddress; /* 入出力データの終了アドレス（メモリ指定）*/
  int  MemoryInitSize; /* 出力データエリア初期サイズ
                                                （メモリ指定）*/
  int  MemoryIncrementSize; /* 出力データエリア増分サイズ
                                                （メモリ指定）*/
} ETTRANSADRSLIST;
```

データ種別 DataKind には，ヘッダファイル ETtrans.h 内で定義されている次の値のどれかを指定します。

- ET_DK_FILE：データ種別がファイルの場合
- ET_DK_MEMORY：データ種別がメモリの場合

ET_DK_MEMORY を指定した場合は，入出力データの開始アドレス DataAddress と終了アドレス EndAddress を指定してください。ただし，出力フォーマットの場合だけ，開始アドレスとして NULL アドレスを指定できます。この場合，トランスレータが必要なデータエリアを確保し，その先頭アドレスを入出力データの開始アドレス DataAddress へ設定して返します。トランスレータが確保したエリアは，次回の ETtransExec の実行，又は ETtransEnd の実行前まで有効です。確保するデータエリアサイズは，pAdrsList の MemoryInitSize，又は MemoryIncrementSize で指定します。pAdrsList の該当する値に 0 が指定された場合は，システム情報ファイル ettrans.ini 内の [MemoryData] セクションで定義された値を用います。pAdrsList，システム情報ファイルの両方に定義がない場合は，初期割り当てサイズと割り当て増分サイズの両方に 1,024（バイト）を仮定します。
出力フォーマットに対しては，入出力データの終了アドレス EndAddress にデータ最終アドレスを返します。ただし，出力データが指定されたアドレスを超えてエ

ラーのために未出力になる場合は，ヘッダファイル「ETtrans.h」で定義されている次の値を設定して返します。

- ET_AD_OVER：メモリアドレスオーバー
- ET_AD_NOTOUT：未出力

ログファイル名 pszLogFile は，ETtransInit 関数で指定したファイルとは別のファイルに出力する場合に指定します。この引数に NULL を指定した場合，又は指定したファイルを作成又は更新できない場合は，ETtransInit 関数で指定したファイルに出力します。ただし，システム情報ファイルの [Option] セクションでログ出力の抑止が指定されている場合は，pszLogFile 指定に関係なく，ログを出力しません。
変換対象指定 iFileAppendFlag は，変換結果をファイルに出力する場合に有効になります（XML 文書の出力を除きます）。ヘッダファイル ETtrans.h 内で定義されている次の値を指定します。

- ET_FA_UPDATE：新規又は上書き
- ET_FA_APPEND：追加書き

変換対象フォーマット指定 iFNFlag に ET_OP_FNOFF を指定した場合は，対象となる MDL ファイル内すべてのフォーマットを変換します。ET_OP_FNON を指定した場合は，差し替え情報 pAdrsList で指定されたフォーマットだけを変換します（コマンド実行時に -FN オプションを指定した場合と同じです）。
エラー情報リスト pErrList へは変換実行時のエラー情報を出力します。エラー情報がない場合は，エラー情報リスト pErrList のデータ数に 0 を返します。取り出した内容は，次回の EttransExec の実行，又は ETtransEnd の実行前まで有効です。構造体 ETTRANSERRLIST は，ヘッダファイル ETtrans.h 内で定義されています。

```
typedef struct {
    int  Size;                  /* エラー情報データ数 */
    ETTRANSERRDATA  *ErrData;   /* エラー情報データ （配列） */
} ETTRANSERRLIST;

typedef struct {
    int  MessageNo;             /* メッセージ番号 */
    int  ErrorLevel;            /* エラーレベル */
    int  NumericData;           /* 数値情報 */
    char  *Information;         /* 文字列情報 */
} ETTRANSERRDATA;
```

各エラー情報のデータは，変換処理で発生したワーニングレベル以上のエラー情報が発生順に設定されます。エラーレベルは，トランスレータの戻り値と同じ意味を持ちます。数値情報と文字列情報は，ログファイルへ出力する付加情報で，次の内容を出力します。付加情報がない場合は，数値情報は 0，文字列情報は NULL が設定されます。

表 D-2　ログファイルに出力される数値情報，文字情報の内容

メッセージ番号	数値情報	文字列情報	備考
9	出力済みのグループ数	なし	グループ単位出力指定がある場合のエラー
55，56	出口関数の戻り値	ユーザ組み込み関数名	出口関数のエラー
7，8，23 ～ 36，50，57，67，78	コンポーネントのアドレス	コンポーネント名	コンポーネントに対するエラー
21	0	オプション	オプション不正のエラー
	1	フォーマット名	
	2	なし（メモリアドレス不正の場合）	

注

メッセージ番号の内容については，「付録 B　トランスレータのメッセージ」を参照してください。

戻り値

トランスレータ起動コマンド実行時の戻り値と同じです。ettrans コマンドの戻り値については，「5.4.2　ettrans コマンドの戻り値」を参照してください。

付録 D.4　ETtransUnLoadMap（MDL ファイルのアンロード）

形式

```
#include  "ETtrans.h"

int  ETtransUnLoadMap (char  *c)
```

引数

引数	種別	内容
c	入力	システムの管理情報のアドレス（ETtransInit 関数の引数 c で取得したアドレス）を指定します。

説明

ETtransLoadMap 関数でロードした MDL ファイルをメモリ上から破棄します。MDL ファイルがロードされていない状態でこの関数を実行するとエラーとなります。

戻り値

戻り値	内容
0x00000000	正常に終了しました。

戻り値	内容
0x04000000	MDL ファイルをメモリ上から破棄できませんでした。

付録 D.5　ETtransEnd（トランスレータの終了）

形式

```
#include  "ETtrans.h"

int  ETtransEnd (char  *c)
```

引数

引数	種別	内容
c	入力	システムの管理情報のアドレス（ETtransInit 関数の引数 c で取得したアドレス）を指定します。

説明

トランスレータを終了します。
MDL ファイルがロードされている状態でこの関数を実行すると，MDL ファイルは自動的にアンロードされます。
データ変換処理を終了するには，この関数を実行して正常に終了したことを確認する必要があります。

戻り値

戻り値	内容
0x00000000	正常に終了しました。
0x04000000	トランスレータの終了処理に失敗しました。

付録 D.6　ETtransInitT（変換処理の初期化）

形式

```
#include  "ETtrans.h"

int  ETtransInitT(char  *c, char  **ct)
```

引数

引数	種別	内容
c	入力	システムの管理情報のアドレス（ETtransInit 関数の引数 c で取得したアドレス）を指定します。
ct	出力	サブシステム管理情報を取得します。

説明

変換処理の初期化を実行します。

変換処理の実行単位でスレッド化する場合，この関数を実行して引数 ct で取得したサブシステム管理情報（アドレス）を ETtransEexeT 関数及び ETtransEndT 関数へ渡す必要があります。

戻り値

戻り値	内容
0x00000000	正常に終了しました。
0x04000000	変換処理の初期化に失敗しました。
0x04000002	メモリ不足です。

付録 D.7　ETtransExecT（変換処理の実行）

形式

```
#include  "ETtrans.h"

int  ETtransExecT(char  *c, char  *ct,
                  ETTRANSADRSLIST  *pAdrsList,
                  char  *pszLogFile,  char  *pszOptList,
                  int  iFileAppendFlag, int  iFNFlag,
                  ETTRANSERRLIST  *pErrList)
```

引数

引数	種別	内容
c	入力	システムの管理情報が格納されているアドレス（ETtransInit 関数の引数 c で取得したアドレス）を指定します。
ct	入力	サブシステムの管理情報が格納されているアドレス（ETtransInitT 関数の引数 ct で取出したアドレス）を指定します。
pAdrsList	入力	変換対象フォーマットの入出力データ差し替え情報の配列を指定します。
pszLogFile	入力	ログファイル名を指定します。
pszOptList	入力	トランスレータ実行時のオプションを指定します。指定できる実行時のオプションについては，「5.4.1(3)　引数」を参照してください。
iFileAppendFlag	入力	ファイルの出力方法を指定します。 • ET_FA_UPDATE：新規又は上書き • ET_FA_APPEND：追加書き
iFNFlag	入力	MDL 内のフォーマットのうち，変換対象とするフォーマットを特定するかどうかを指定します。 • ET_OP_FNOFF：指定なし • ET_OP_FNON ：指定あり
pErrList	出力	エラー情報リストを取得します。

説明

指定されたオプションでトランスレータを起動し，ETtransLoadMap 関数でロード

したMDLファイルに定義されている内容に従ってフォーマットのデータを変換します。
詳細については，「付録 D.3　ETtransExec（トランスレータの起動）」を参照してください。ただし，トランスレータが確保した出力フォーマットのメモリエリア及びエラー情報リストの内容は，同一スレッド内で，次の ETtransExecT 関数又は ETtransEndT 関数を実行するまで有効です。

戻り値

ettrans 実行時の戻り値と同じです。戻り値については，「5.4.2　ettrans コマンドの戻り値」を参照してください。

付録 D.8　ETtransEndT（変換処理の終了）

形式

```
#include  "ETtrans.h"

int  ETtransEndT(char  *c, char  *ct)
```

引数

引数	種別	内容
c	入力	システムの管理情報のアドレス（ETtransInit 関数の引数 c で取得したアドレス）を指定します。
ct	入力	サブシステムの管理情報のアドレス（ETtransInitT 関数の引数 ct で取得したアドレス）を指定します。

説明

変換処理を終了します。
変換処理の実行単位でスレッド化する場合，この関数を実行して変換処理が正常に終了したことを確認してください。

戻り値

戻り値	内容
0x00000000	正常に終了しました。
0x04000000	変換処理の終了に失敗しました。

付録 D.9　互換 API の使用例

互換 API の使用例を，次に示します。

```
#include <stdio.h>
#include "ETtrans.h"

static void print_message(int iCode, char *msg,
                          ETTRANSERRLIST *ErrList);
```

```
int main()
{
  int iRet = 0;
  ETTRANSADRSLIST AdrsList[3];
  ETTRANSERRLIST ErrList;
  char *c;
                  /* アドレスリストの設定 */
  AdrsList[0].FormatName = "IN";
  AdrsList[0].DataKind = ET_DK_FILE;
  AdrsList[0].DataAddress = "input.txt";
  AdrsList[1].FormatName = "OUT";
  AdrsList[1].DataKind = ET_DK_FILE;
  AdrsList[1].DataAddress = "output.txt";
  AdrsList[2].FormatName = NULL;
                  /* トランスレータ初期化 */
  iRet = ETtransInit(&c, NULL);
  if (iRet) {
    print_message(iRet, "ETtransInit", NULL);
    goto EXIT;
 }
                  /* MDLローディング */
  iRet = ETtransLoadMap(c, "SAMPLE.mdl");
  if (iRet) {
    print_message(iRet, "ETtransLoadMap", NULL);
    goto END;
 }
                  /* 変換実行 */
  iRet = ETtransExec(c, AdrsList, NULL, "-IERR",
                     ET_FA_UPDATE, ET_OP_FNOFF, &ErrList);
  if (iRet) {
    print_message(iRet, "ETtransExec", &ErrList);
    goto END;
 }
                  /* MDLアンロード */
  iRet = ETtransUnLoadMap(c);
  if (iRet) {
    print_message(iRet, "ETtransUnLoadMap", NULL);
    goto END;
 }

END:              /* 終了処理 */
  iRet = ETtransEnd(c);
  if (iRet) {
    print_message(iRet, "ETtransEnd", NULL);
 }

EXIT:
  printf("--> Return Code = 0x%08x¥n", iRet);
  return iRet;
 }

/*
 * エラー情報出力関数
 */
static void print_message(int iCode, char *msg,
                          ETTRANSERRLIST *ErrList)
 {
```

```
  int i;
  ETTRANSERRDATA *edata;

  printf("### ERROR %s %d ###¥n", msg, iCode);
  if (ErrList) {
    printf(" Error Size = %d¥n", ErrList->Size);
    for (i = 0; i < ErrList->Size; i++) {
       edata = &ErrList->ErrData[i];
       printf("¥t%02d (0x%08x) num=%d",
         edata->MessageNo, edata->ErrorLevel, edata->NumericData);
       if (edata->Information) {
         printf(" info=%s¥n", edata->Information);
       } else {
         printf("¥n");
       }
    }
  }
}
```

付録 E　Interschema から出力される情報

Interschema から出力される情報について説明します。

付録 E.1　マップ式ファイル

マップ式ファイルは，マップ式をテキスト形式で記述したファイルです。対象となるフォーマットのルートコンポーネントから順番に，すべてのコンポーネントのマップ式や変数式などの情報を記述します。マップ式ファイルは，次のどちらかの方法で出力します。

- マージツールのコマンド「etmerge」
- MDL エディタの外部出力コマンド又はマージコマンド

出力したマップ式は，マップ式ファイル上で直接編集できます。マップ式ファイルの編集内容は，次のどちらかの方法で MDL ファイルに反映できます。

- マージツールのコマンド「etmerge」
- MDL エディタのマージコマンド

(1) 記述形式

マップ式ファイルのフォーマットの記述形式を次に示します。なお，マップ式ファイルは項目ごとに改行されます。

表 E-1　マップ式ファイルのフォーマットの記述形式

項目	説明
コンポーネントのグローバルパス名	コンポーネントのグローバルパス名です。
マップ式	マップ式文字列です。マップ式がない場合は空行になります。
%NOUSE%	未使用のコンポーネントを表します。
%IOCHANGE%n	グループ単位出力指定があることを表します。n はラベル「1」～「255」の数値です。
%ORDER%	MDL エディタで編集した順序決定式があることを表します。選択構造コンポーネントの場合だけ出力されます。
順序決定式	順序決定式文字列です。MDL エディタで式を削除した場合は出力されません。
%CASE%% 定数種別 %	MDL エディタで編集した順序決定値があることを表します。選択構造の子コンポーネントの場合だけ出力されます。 定数種別には，次のどれかが出力されます。 • NUM：数値 • STRING：文字列 • STREAM：バイト列

項目	説明
順序決定値	順序決定値です。定数種別の値が次の形式で出力されます。 • 数値：10 進数表記の文字列 • 文字列 • バイト列：0x 付きの 16 進文字列 ただし，MDL エディタで値を削除した場合は出力されません。
%DEFAULT%	MDL エディタで編集したデフォルト式があることを表します。コンポーネントが型の場合だけ出力されます。
デフォルト式	デフォルト式文字列です。MDL エディタで式を削除した場合は出力されません。
%CONDITION%	MDL エディタで編集したコンポーネントの条件式があることを表します。
条件式	条件式文字列です。MDL エディタで式を削除した場合は出力されません。
%DEPEND%	MDL エディタで編集した出現回数決定式があることを表します。
出現回数決定式	出現回数決定式文字列です。MDL エディタで式を削除した場合は出力されません。
%SIZE%	MDL エディタで編集したサイズ決定式があることを表します。
サイズ決定式	サイズ決定式文字列です。MDL エディタで式を削除した場合は出力されません。
%%-%%	変数式の区切りを表す記号文字列（固定文字列）です。変数ごとに，この区切りと変数式が繰り返されます。
変数式	1 変数分の変数式文字列です。
%-%	1 コンポーネントの終了を表す記号文字列（固定文字列）です。

(2) 注意事項

マップ式ファイルの注意事項を次に示します。

- マージ時にコンポーネント名が重複する場合は，最後に出現したものを優先します。
- 「%NOUSE%」指定は，MDL エディタの［コンポーネントのプロパティ］ダイアログで「このコンポーネントを使用しない」を選択して未使用の定義をするのと同じ意味になります。また，「%IOCHANGE%」指定は，［コンポーネントのプロパティ］ダイアログで「グループ単位出力指定」と「ラベル」を指定するのと同じ意味になります。
- 式文字列及び定数値以外の部分は正当性を検証しないので，直接編集しないでください。
- 変数式は，定義した変数ごとに区切って出力されます。
- マージツール「etmerge」のマッピング機能，又は MDL エディタのマージコマンドのマップ式マージで，表 E-1 の「%ORDER%」～「サイズ決定式」の情報がマージされた場合，MDL 上で式又は値が編集されたものと見なされます。
- MDL 上で式又は値を編集した後にフォーマットをマージした場合，マージ元の式又は値と，マージ先のフォーマットの式又は値が同じときは，MDL 上での編集は，な

しの状態に戻ります。式又は値が同じかどうかは，その文字列の単純比較で判定します。

(3) 記述例

マップ式ファイルの記述例を次に示します。

```
OUT@ROOT@STRUCT
= IN@ROOT@MESSAGE[INDEX(0)];
%IOCHANGE%1
%ORDER%
OUT@v1
%%-%%
OUT@v1 = OUT@v1 + 1;
%%-%%
OUT@v2 = OUT@V1 * 10;
%-%
OUT@ROOT@STRUCT@data0

%NOUSE%
%-%
OUT@ROOT@STRUCT@data1
= $@data1;
%CASE%%NUM%
1
%CONDITION%
# > 0
%-%
OUT@ROOT@STRUCT@data2
= $@data2 + IN@ROOT@HEAD@num;
%CASE%%STRING%
N2
%DEFAULT%
= 0;
%SIZE%
10
%-%
OUT@ROOT@STRUCT@data3
= $@data3[INDEX(0)];
%CASE%%STREAM%
0x03
%DEPEND%
OUT@v2
%-%
```

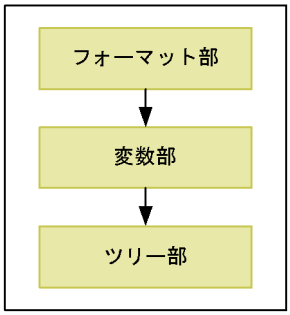
付録 E.2　ツリー情報ファイル

ツリー情報ファイルは，FDL 又は MDL の内容をツリー形式で出力し，各種属性の情報を記述したファイルです。ツリー情報ファイルは，FDL エディタ又は MLD エディタの外部出力コマンドで出力します。FDL エディタの場合は，フォーマット情報ファイルの一部として出力されます。MDL エディタの場合は，フォーマット情報ファイルの一部として出力するか，ツリー情報ファイルだけ出力するかを指定できます。フォーマット情報ファイルについては，「付録 E.3　フォーマット情報ファイル」を参照してください。

(1) 記述形式

ツリー情報ファイルは，FDL 又は MDL のフォーマット単位に記述されます。1 フォーマット分の記述形式を次に示します。

図 E-1　ツリー情報ファイルの記述形式



フォーマット部には，フォーマット名とそのフォーマットの属性が出力されます。

変数部には，1 行に 1 変数の情報が，変数の数だけ出力されます。

ツリー部には，1 行に 1 コンポーネントのコンポーネント情報が出力されます。ツリーには，MDL エディタと同様，ルート構造以下が表示されます。FDL のフォーマットの場合，ルート構造からつながっていない型や構造などは出力されません。

(2) フォーマット部

フォーマット部の記述形式を次に示します。

```
フォーマット名（TAB）フォーマット属性
```

（凡例）
　（TAB）：タブを表します。

フォーマット属性として出力される項目を，出力される順番に次に示します。なお，項目間はスペースで区切られます。

表 E-2　フォーマット属性として出力される項目

項目	出力内容	説明	備考
入出力種別	(i)	入力専用	入出力兼用の場合は，出力されません。
	(o)	出力専用	

項目	出力内容	説明	備考
規格種別	CII	―	規格種別が「なし」の場合は，出力されません。
	CII3		
	EDIFACT		
	EDIFACT4		
	SNACCS	Sea-NACCS（旧）	
	SNACCS2	Sea-NACCS	
	SNEDIFACT	Sea-NACCS 用 EDIFACT（旧）	
	SNEDIFACT2	Sea-NACCS 用 EDIFACT	
	XMLDTD	XML	
	CSV	―	
データ形式	(big)	ビッグエンディアン	―
	(little)	リトルエンディアン	
文字コード	1 バイト文字列：2 バイト文字列：混在文字列：数値：セパレータ	―	各文字コードは，コード指定子です。コード指定子については，「6.3.1　トランスレータが扱う文字コード」を参照してください。
サイズ系指定	(byte)	バイト数	―
	(char)	文字数	
不正文字コード置換	(spron)	ON	―
	(sproff)	OFF	
溢れカット	(cuton)	ON	―
	(cutoff)	OFF	
不要文字削除	(delon)	ON	―
	(deloff)	OFF	

（凡例）

―：特にありません。

(3) 変数部

フォーマット部の記述形式を次に示します。

```
（indent）@ 変数名（TAB）初期値
```

（凡例）

（indent）：インデントを表します。1 階層分のインデントは，スペース 2 文字分です。

（TAB）：タブを表します。

(4) ツリー部

ツリー部の記述形式を次に示します。

- 構造のコンポーネントの場合

```
（indent）+ 構造のコンポーネント名[補助情報] [出現数]（TAB）構造属性（TAB）コンポーネント属性
```

- 型のコンポーネントの場合

```
（indent）+ 型のコンポーネント名[補助情報] [出現数]（TAB）型属性（TAB）コンポーネント属性
```

- XML 属性の場合

```
（indent）+ @ XML属性名（TAB）XML属性の属性（TAB）コンポーネント属性
```

（凡例）

（indent）：インデントを表します。1 階層分のインデントは，スペース 2 文字分です。最上位構造（ルート）は 1 階層分，その他は階層数分のインデントです。

（TAB）：タブを表します。

[補助情報]：次の条件に該当する場合だけ，次の内容が出力されます。

選択構造の場合：(P)

未使用指定がある場合：(NOUSE)

[出現数]：次の内容が出力されます。ただし，1 回固定のコンポーネントに対しては出力されません。n は最小，m は最大の意味です。

固定の場合：[n]

可変の場合：[n:m]

可変（無限大）の場合：[n:]

動的指定ある場合：[出現数](dep)

属性として出力される項目を，出力される順番に次に示します。なお，項目間は基本的にスペースで区切られますが，複数の項目を連続して記述する場合もあります。また，定義されていない項目，及び FDL エディタで「フォーマットの指定に従う」を指定した項目は出力されません。

表 E-3　構造属性として出力される項目

項目	出力内容	説明	備考
セパレータ	<SEPA> (b) 開始文字 (p) 中間区切り（要素前）(i) 中間区切り（要素間）(a) 終了文字 (r) 解放文字	開始文字，中間区切り，終了文字，及び解放文字は，「値」又は「セパレータ名 (値)」です。(値) はそのセパレータの値です。	セパレータを定義した場合だけ出力されます。予約セパレータの場合，(値) は出力されません。
順序決定式	<ORDER> 式文字列	－	式文字列中のスペース，タブ，及び改行コードは削除されます。ただし，文字列定数部は除きます。
条件式	<CHECK> 式文字列	－	式文字列中のスペース，タブ，及び改行コードは削除されます。ただし，文字列定数部は除きます。
規格情報（CII3）	<<MSG>>BPID INFO	メッセージ構造 • BPID：サブ機関，版を含む BPID です（8 桁）。 • INFO：情報区分です（4 桁）。	メッセージ構造及びマルチ明細構造以外は出力されません。
	<<MULTI>>NO	マルチ明細構造 • NO：マルチ明細番号です。拡張マルチは「0x」が付いた 16 進文字列，その他は文字です。	
規格情報（EDIFACT4）	<<SEG>>	Segment	－
	<<CMP>>	Component（S001，S004 を含む）	
	<<REP>>	Repetition	
	<<UNA>>	UNA Segment	

（凡例）

－：特にありません。

表 E-4　型属性として出力される項目

項目	出力内容	説明	備考
主属性	int	整数	―
	num	実数	
	imp	暗黙的小数部付数値	
	fd	ゾーン形式数値	
	fdp	パック形式数値	
	s1	符号付 2 進整数（1 バイト）	
	s2	符号付 2 進整数（2 バイト）	
	s3	符号付 2 進整数（3 バイト）	
	s4	符号付 2 進整数（4 バイト）	
	us1	符号無 2 進整数（1 バイト）	
	us2	符号無 2 進整数（2 バイト）	
	us3	符号無 2 進整数（3 バイト）	
	us4	符号無 2 進整数（4 バイト）	
	str1	1 バイト文字列	
	str2	2 バイト文字列	
	str	混在文字列	
	stm	バイト列	
	date	日付	
	time	時刻	
	datetime	日付時刻	
サイズ	n	固定 n：最小	主属性が int，num，imp，str1，str2，str，stm，date，time，又は datetime の場合だけ出力されます。
	n:m	可変 n：最小 m：最大（省略時は出力されません）	
	n:m(dep)	動的指定 n：最小 m：最大（省略時は出力されません）	
サイズ系指定	(byte)	バイト数	主属性が str1，str2，str，date，time，又は datetime の場合だけ出力されます。
	(char)	文字数	

項目	出力内容	説明	備考
文字コード	コード指定子	―	主属性が int，num，imp，fd，str1，str2，str，date，time，又は datetime の場合だけ出力されます。 コード指定子については，「6.3.1　トランスレータが扱う文字コード」を参照してください。
数値桁数 符号 指数部 桁セパレータ 符号カウント指定	(place)(1)(2)(3)(4)(5)(6)(7)(8)(9) (10)	(1)：先頭符号（-：負だけ，*：常にあり） (2)：桁数（全体） (3)：小数点文字「.」 (4)：桁数（小数部） (5)：末尾符号（-：負だけ，*：常にあり） (6)：指数記号 (7)：指数部符号（-：負だけ，*：常にあり） (8)：桁数（指数部） (9)：桁セパレータ (10)：符号カウント指定「(c)」	主属性が int，num，imp，fd，又は fdp の場合だけ出力されます。 (1) ～ (10) に該当する内容がない場合は出力されません。
日付／時刻書式	YYYYMMDD	8 桁の日付表示	主属性が date 又は time の場合だけ出力されます。
	YYMMDD	6 桁の日付表示	
	HHMMSS	6 桁の時刻表示	
日付時刻書式	(1)(2)(3)	(1)：先頭文字列 (2)：パート書式 CCYY：西暦年 CC：西暦年上位 2 桁 YY：年（西暦年下 2 桁） MM：月 DD：日 hh：時 mm：分 ss：秒 ZZ：ゾーン部 (3)：後続文字列	主属性が datetime の場合だけ出力されます。

項目	出力内容	説明	備考
桁寄せ 埋め字 不要文字削除	(1)(2)(3)	(1)：桁寄せ (l)：左寄せ (r)：右寄せ (2)：埋め字の値 (3)：不要文字削除 (delon)：ON (deloff)：OFF (delon_poff)：全体 ON,パート値 OFF (deloff_pon)：全体 OFF,パート値 ON	主属性が int，num，imp，str1，str2，str，date，time，又は datetime の場合だけ出力されます。 (1) は，文字列／日時が左寄せ，又は数値が右寄せの場合は出力されません。 (2) は，文字列／日時がスペース，又は数値が 0 の場合は出力されません，ただし，バイト列定義の場合は出力されます。
値定義	<VALUE>(nochk)	—	(nochk) は「他の値が現れてもエラーにしない」を指定した場合だけ出力されます。 個々の値は出力されません。
規格情報 (CII3)	<<TFD>>(1)(2)	(1)：タグ番号 (2)：属性種別 (9)：9 属性 (N)：N 属性 (Y)：Y 属性 (X)：X 属性 (K)：K 属性 (B)：B 属性	TFD 以外は出力されません。

（凡例）

—：特にありません。

表 E-5　XML 属性として出力される項目

項目	出力内容	説明	備考
属性型	CDATA	—	—
	ID		
	IDREF		
	IDREFS		
	NMTOKEN		
	NMTOKENS		
	ENTITY		
	ENTITIES		
	NOTATION		

項目	出力内容	説明	備考
	ENUMERATION	列挙型	
デフォルト値タイプ	#req	#REQUIRED	－
	#imp	#IMPLIED	
	#fix	#FIXED	
	#val	デフォルト値	
列挙値	<VALUE>	－	個々の値は出力されません。
デフォルト値	<DEFAULT> 値	－	－

（凡例）

－：特にありません。

表 E-6　コンポーネント属性として出力される項目

項目	出力内容	説明	備考
グループ単位出力指定	(io)n	n：ラベル	－
カウンタ	(clear)	初期化指定	－
	(increment)	インクリメント指定	
順序決定値	<CASE> 値	－	－
出現回数決定式	<EXISTCOUNT> 式文字列	－	式文字列中のスペース，タブ，及び改行コードは削除されます。ただし，文字列定数部は除きます。
条件式	<CHECK> 式文字列	－	式文字列中のスペース，タブ，及び改行コードは削除されます。ただし，文字列定数部は除きます。
トランスレータ指示	<PRAGMA> 定義文字列	－	この情報は辞書で設定される情報です。FDL エディタ及び MDL エディタでは操作できません。
サイズ決定式	<SIZE> 式文字列	－	式文字列中のスペース，タブ，及び改行コードは削除されます。ただし，文字列定数部は除きます。
デフォルト式	<DEFAULT> 式文字列	－	式文字列中のスペース，タブ，及び改行コードは削除されます。ただし，文字列定数部は除きます。

項目	出力内容	説明	備考
マップ式	<MAP> 式文字列	－	文字列定数部，及び %delay 部を除く式文字列中のスペース，タブ，及び改行コードは削除されます。
子マップ式	<CHILDMAP> 式文字列	－	文字列定数部，及び %delay 部を除く式文字列中のスペース，タブ，及び改行コードは削除されます。 複数式ある場合は，次のように式と式の間にスペース文字が入ります。 ＜ xxxxx> 式文字列 1 式文字列 2 式文字列 3… なお，この情報は辞書で設定される情報です。FDL エディタ及び MDL エディタでは操作できません。
変数式	<VAR> 式文字列	－	式文字列中のスペース，タブ，及び改行コードは削除されます。ただし，文字列定数部は除きます。 複数式ある場合は，次のように式と式の間にスペース文字が入ります。 ＜ xxxxx> 式文字列 1 式文字列 2 式文字列 3・・・

（凡例）

－：特にありません。

(5) 値

各属性の「値」（定数値）は，次のように出力されます。

数値：浮動小数形式

例　1.5

文字列：ダブルクォーテーション囲み

例　"abc"

バイト列：「0x」付き 16 進文字列

例　x0d0a

(6) 記述例

ツリー情報ファイルの記述例を次に示します。

```
UXML                (i) XMLDTD (big) JIS8:JISK:UTF16:JIS8:JIS8 (char) (spron) (cuton) (deloff)
  + ユーザ登録票
    + @ 登録形態      ENUMERATION #req <VALUE>
    + 商品名          str 0:
    + 型番            str 0:
    + シリアル番号    str 0:
    + ユーザID番号    str 0:
    + お名前
      + ふりがな
        + 姓          str 0:
        + 名          str 0:
      + 漢字
        + 姓          str 0:
        + 名          str 0:
    + 生年月日        str 0:
    + 性別            str 0:
    + 住所
      + 郵便番号      str 0:
      + 住所番地      str 0:
    + 'ユーザ登録票#1' [0:1]
      + 法人名        str 0:
      + 所属部課      str 0:
    + 'e-mail'        str 0:
      + @ DM許容      ENUMERATION #val <VALUE> <DEFAULT>"Y"
    + 購入店名        str 0:
    + 購入日          str 0:
    + 意見感想        str 0:

IN                  (i) (big) JIS8:JISK:SJIS:JIS8:JIS8 (byte) (spron) (cuton) (delon)
  + ROOT
    + RECORD [1:]
      + 製品コード    str1 20
      + 単価          int 7 (place)7
      + 個数          int 5 (place)5

OUT                 (o) (big) JIS8:JISK:SJIS:JIS8:JIS8 (byte) (spron) (cuton) (delon)
  @ v1              0
  + ROOT
    + RECORD [1:]   <SEPA> (i)SEPA1(",") (a)SEPA2(0x0d0a)  <MAP>=IN@ROOT@RECORD[INDEX(0)];
      + 製品コード  str1 1:20               <MAP>=$@製品コード;
      + 希望価格    int 1: (place)7         <MAP>=$@単価;
      + 数量        int 1: (place)5         <MAP>=$@個数;
      + 小計金額    int 1: (place)15        <MAP>=$@単価*$@個数; <VAR>OUT@v1=OUT@v1+#;
```

付録 E.3　フォーマット情報ファイル

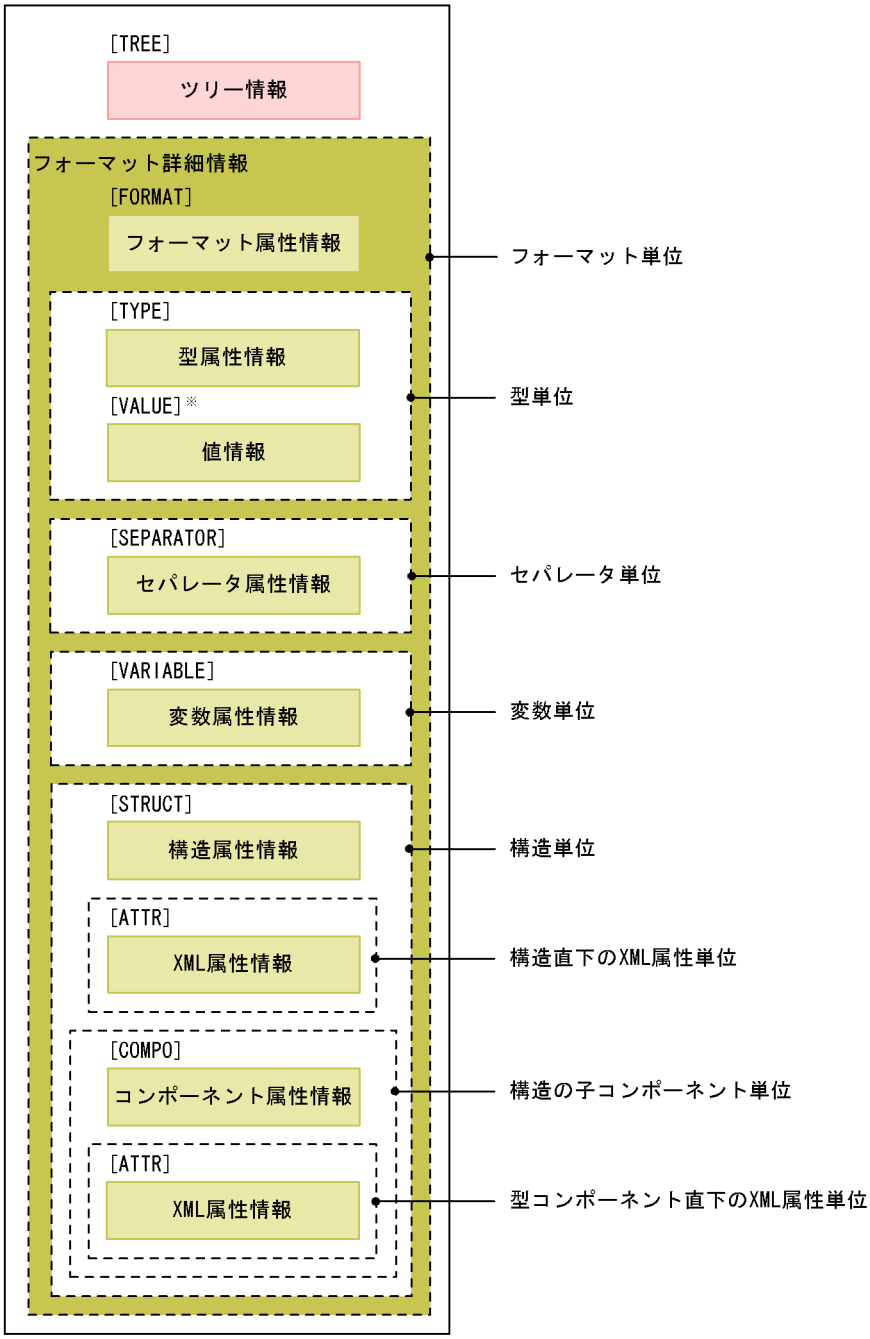
フォーマット情報ファイルは，FDL 又は MDL のプロパティ情報を記述したファイルです。フォーマット情報ファイルは，FDL エディタ又は MLD エディタの外部出力コマンドで出力します。

(1) 記述形式

フォーマット情報ファイルは，ツリー情報ファイルの内容とフォーマット詳細情報の内

容で構成されています。ツリー情報ファイルについては,「付録 E.2　ツリー情報ファイル」を参照してください。フォーマット詳細情報は，フォーマット単位に記述されます。フォーマット情報ファイルの記述形式を次に示します。

図 E-2　フォーマット情報ファイルの記述形式



（凡例）

：繰り返し出力される情報です。

注※　値定義がある場合だけ出力されます。

各情報の出力順序について次に説明します。

型
　FDL の場合は型定義フォルダの順序，MDL の場合は型名でソートした順序で出力されます。

セパレータ
　FDL の場合はセパレータ定義フォルダの順序，MDL の場合は定義されている順序で出力されます。

変数
　FDL の場合は変数定義フォルダの順序，MDL の場合は定義されている順序で出力されます。

構造
　FDL の場合は構造定義フォルダの順序，MDL の場合は構造名でソートした順序で出力されます。MDL の場合，複数の箇所で使用されている構造は，代表の構造についてだけ出力されます。

コンポーネント
　1 構造に対して，兄弟の順序で出力されます。

XML 属性
　構造直下のものは構造のコンポーネント情報の前に，型直下のものは型コンポーネント情報に続けて，定義された順序で出力されます。

! 注意事項

MDL の場合，構造以下のフォーマット詳細情報は，代表の構造についてだけ出力されます。したがって，MDL 上で編集した式は出力されないものがあります。個々の式については，ツリー情報ファイルで確認してください。

(2) フォーマット詳細情報

次に示す情報の項目タイトル及び項目値が，フォーマット詳細情報として出力されます。

- フォーマット属性情報
- 型属性情報
- セパレータ属性情報
- 変数属性情報
- 構造属性情報
- XML 属性情報
- コンポーネント属性情報

フォーマット詳細情報の記述形式を次に示します。

- 項目値だけ出力される場合

```
項目タイトル（TAB）項目値
```

- 項目値以外の情報も出力される場合

```
項目タイトル（TAB）項目値（TAB）補助情報
```

（凡例）

（TAB）：タブを表します。

注意事項

式文字列に改行が含まれる場合，改行コードはスペースに置き換えられます。

（3） 記述例

フォーマット詳細情報の記述例を次に示します。なお，記述例は実際の出力情報を抜粋したものです。

```
[FORMAT]                ============================================================
ﾌｫｰﾏｯﾄ名                OUT
ﾃﾞｰﾀﾌｧｲﾙ名              d:¥interschema2¥sample¥data¥output.txt
規格指定
入出力種別              出力専用
ﾃﾞｰﾀ格納形式            ﾋﾞｯｸﾞｴﾝﾃﾞｨｱﾝ
ｻｲｽﾞ指定方法            ﾊﾞｲﾄ数
不正文字ｺｰﾄﾞ置換指定    ON
ﾃﾞｰﾀ切り捨て指定        ON
不要文字削除            ON
1ﾊﾞｲﾄ文字列文字ｺｰﾄﾞ      JIS8
2ﾊﾞｲﾄ文字列文字ｺｰﾄﾞ      JISK
混在文字列文字ｺｰﾄﾞ      SJIS
数値文字ｺｰﾄﾞ            JIS8
ｾﾊﾟﾚｰﾀ文字ｺｰﾄﾞ          JIS8
ｺﾒﾝﾄ                    帳票OUTフォーマット

[TYPE]
型名                    希望価格
主属性                  整数

文字ｺｰﾄﾞ                ﾌｫｰﾏｯﾄの指定に従う
左右寄せ                右寄せ
埋め字                  0
ｻｲｽﾞ                    1:
桁数                    全体 7
桁区切り文字
符号                    なし
不要文字削除            ﾌｫｰﾏｯﾄの指定に従う
ｺﾒﾝﾄ
                        :

[SEPARATOR]
ｾﾊﾟﾚｰﾀ名                SEPA1
属性                    文字列
値                      ,
ｺﾒﾝﾄ
                        :

[STRUCT]                ------------------------------------------------------------
構造名                  RECORD
種類                    逐次
中間区切り文字          (要素間) SEPA1 (",")
終了文字                SEPA2 (0x0d0a)
ｺﾒﾝﾄ                    レコード

[COMPO]
ｺﾝﾎﾟｰﾈﾝﾄ名              製品コード
親名                    RECORD
出現数                  [1]
必須指定                OFF

ﾏｯﾌﾟ式                  = $@製品コード;
                         :
```

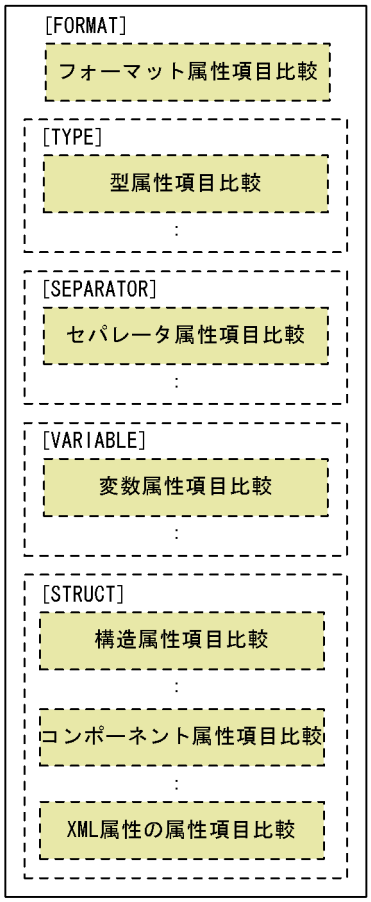
付録 E.4　差分情報ファイル

差分情報ファイルは，FDL フォーマット又は DTD フォーマットのマージ前後の差分情報を記述したファイルです。差分情報ファイルは，MDL エディタのマージコマンドで出力します。

(1) 記述形式

差分情報ファイルの記述形式を次に示します。

図 E-3　差分情報ファイルの記述形式



（凡例）

：一部の情報が繰り返し出力されます。

(a) フォーマット属性項目の比較（FORMAT）

［FORMAT］以下に出力される項目を次に示します。

```
-------------------------------------------------
 [FORMAT]
-------------------------------------------------
フォーマット属性項目比較          ･･･ 差分があるすべての項目を出力
```

(b) 型属性項目の比較（TYPE）

［TYPE］以下に出力される項目を次に示します。

```
-------------------------------------------------
 [TYPE]
-------------------------------------------------
<<CHANGE>> 型名                   ･･･ 差分があるすべての型を型単位で出力
型属性項目比較

<<ADD>>                           ･･･ 追加された型名だけを出力
 (TAB) 型名

<<DEL>>                           ･･･ 削除された型名だけを出力
 (TAB) 型名
```

（凡例）
（TAB）：タブを表します。

(c) セパレータ属性項目の比較（SEPARATOR）

［SEPARATOR］以下に出力される項目は，型情報の「型」を「セパレータ」に置き換えた内容と同じです。

(d) 変数属性項目の比較（VARIABLE）

［VARIABLE］以下に出力される項目は，型情報の「型」を「変数」に置き換えた内容と同じです。

(e) 構造属性項目，コンポーネント属性項目および XML 属性項目の比較（STRUCT）

［STRUCT］以下に出力される項目を次に示します。

```
------------------------------------------------------
 [STRUCT]
------------------------------------------------------
<<CHANGE>> 構造コンポーネントグローバル名
                                ･･･ 差分がある構造コンポーネント単位で出力
構造属性項目比較                ･･･ 差分があるすべての項目を出力

COMPONENT LIST:                 ･･･ 子コンポーネント定義部
 (TAB) <<CHANGE>>
 (TAB) (TAB) 子コンポーネント名 (TAB) [b] 順序番号 (TAB) [a] 順序番号
                                ･･･ 兄弟位置が変更された場合に出力
 (TAB) <<ADD>>                  ･･･ 追加されたコンポーネント名だけを出力
 (TAB) (TAB) 子コンポーネント名
 (TAB) <<DEL>>                  ･･･ 削除されたコンポーネント名だけを出力
 (TAB) (TAB) 子コンポーネント名

XML属性リスト比較※             ･･･ ルートコンポーネントの場合に出力

<COMPONENT> 子コンポーネント名
                                ･･･ 差分がある子コンポーネント単位で出力
コンポーネント属性項目比較      ･･･ 差分があるすべての項目を出力

XML ATTR LIST:                  ･･･ XML属性定義部
 (TAB) <<CHANGE>>
 (TAB) (TAB) XML属性名 (TAB) [b] 順序番号 (TAB) [a] 順序番号
                                ･･･ 兄弟位置が変更された場合に出力
 (TAB) <<ADD>>                  ･･･ 追加されたXML属性名だけを出力
 (TAB) (TAB) XML属性名
 (TAB) <<DEL>>                  ･･･ 削除されたXML属性名だけを出力
 (TAB) (TAB) XML属性名

<COMPONENT> 子コンポーネント名 <XML ATTR> XML属性名
                                ･･･ 差分があるXML属性単位で出力
XML属性属性項目比較             ･･･ 差分があるすべての項目を出力

------------------------------------------------------
```

（凡例）
　(TAB) ：タブを表します。
　[b]：変更前を表します。
　[a]：変更後を表します。
　順序番号：0から始まる整数です。

注※
　XML属性群に対して行う差分比較です。
　上記出力項目の「XML ATTR LIST」～「XML属性属性項目比較」のことです。

(f)　注意事項

差分情報ファイルに出力される各情報の注意事項を次に示します。

- 一つの構造に対する比較は，グローバル名が一致するコンポーネント同士が比較され，差分が出力されます。ただし，ルートは名称に関係なく必ず比較されます。また，MDL エディタのマージコマンドでフォーマット名だけ変更したコンポーネント同士は，比較の対象となります。

- グローバル名が一致するコンポーネントでも，型と構造の種別を変更した場合は，構造属性の比較はされません。
- XML 属性は，構造の子コンポーネントの下で比較されます。ただし，ルート構造直下の XML 属性だけ，構造の下（子コンポーネントの比較前）で比較されます。
- 構造の子コンポーネント及び XML 属性の定義情報（兄弟順序や定義の有無）の差分は，リスト情報として変更位置，追加，及び削除の情報が出力されます。
- 型，セパレータ，及び変数は，同名の要素同士が比較されます。追加又は削除された要素に対しては，名称が出力されます。

(2) 属性項目比較の記述形式

各属性項目の差分情報の記述形式を次に示します。なお，差分がない場合は出力されません。

```
項目タイトル:
（TAB）[b] 変更前の内容
（TAB）[a] 変更後の内容
```

（凡例）

（TAB）：タブを表します。

[b]：変更前を表します。

[a]：変更後を表します。

(3) 値定義比較／列挙値比較の記述形式

値定義比較／列挙値比較の記述形式を次に示します。マージ前後の両方のフォーマットで値が定義されている場合に出力されます。

```
項目タイトル:
（TAB）[b] 値エラーチェック解除指定※
（TAB）[a] 値エラーチェック解除指定※
（TAB）<<CHANGE>> 値          ･･･ コメントが変更された値単位で出力
（TAB）（TAB）[b](ｺﾒﾝﾄ) コメント文字列
（TAB）（TAB）[a](ｺﾒﾝﾄ) コメント文字列
（TAB）<<ADD>>                ･･･ 追加された値だけを出力
（TAB）（TAB）値
（TAB）<<DEL>>                ･･･ 削除された値だけを出力
（TAB）（TAB）値
```

（凡例）

（TAB）：タブを表します。

[b]：変更前を表します。

[a]：変更後を表します。

注※

値エラーチェック解除指定は，［値定義］ダイアログで「他の値が現れてもエラーにしない」を選択した場合に記述されます。選択しない場合は空欄になります。

(4) 記述例

差分情報ファイルの記述例を次に示します。

```
------------------------------------------------------------
 [FORMAT]
------------------------------------------------------------
2ﾊﾞｲﾄ文字列文字ｺｰﾄﾞ:
        [b] JISK
        [a] SJISK

------------------------------------------------------------
 [TYPE]
------------------------------------------------------------
<<ADD>>
        合計金額

<<DEL>>
        小計金額

------------------------------------------------------------
 [SEPARATOR]
------------------------------------------------------------

<<CHANGE>> SEPA1
値:
        [b] ","
        [a] ";"
ｺﾒﾝﾄ:
        [b]
        [a] セミコロン

------------------------------------------------------------
 [VARIABLE]
------------------------------------------------------------
<<ADD>>
        var
```

```
------------------------------------------------------------
 [STRUCT]
------------------------------------------------------------
<<CHANGE>> OUT@ROOT
COMPONENT LIST:
        <<ADD>>
                TRAILER

<COMPONENT> RECORD
変数式:
        [b]
        [a] OUT@var = OUT@var + #@希望価格 * #@数量;

------------------------------------------------------------
<<CHANGE>> OUT@ROOT@RECORD
開始文字:
        [b]
        [a] "D"
中間区切り文字:
        [b] (要素間) SEPA1 (",")
        [a] (要素前) SEPA1 (";")

COMPONENT LIST:
        <<DEL>>
                小計金額

<COMPONENT> 製品コード
出現数:
        [b] [1]
        [a] [2]

<COMPONENT> 数量
条件式:
        [b]
        [a] # > 0

------------------------------------------------------------
```

付録 F　ファイルのバージョンとトランスレータのサポート範囲

FDL ファイル及び MDL ファイルのバージョンと，項目長やデータサイズなどに関するトランスレータのサポート範囲について説明します。

付録 F.1　ファイルのバージョンと Interschema のバージョンとの対応

FDL ファイル及び MDL ファイルは，ファイルの作成に使用したエディタによってバージョンが異なります。ここでは，ファイルのバージョンと Interschema のバージョンとの対応について説明します。なお，FDL ファイル及び MDL ファイルは上位互換です。例えば，電子データ交換ツールで作成した FDL ファイル及び MDL ファイルは，Interschema で編集できますが，Interschema で作成した FDL ファイル及び MDL ファイルを電子データ交換ツールで編集することはできません。

表 F-1　FDL ファイル及び MDL ファイルのバージョン一覧

ファイルのバージョン	Interschema のバージョン	説明
1	―	電子データ交換ツールで作成されたファイル。 初期のバージョン。
2	―	電子データ交換ツールで作成されたファイル。 Sea-NACCS フォーマット（旧）を含む。
3	―	電子データ交換ツールで作成されたファイル。 RDB 接続フォーマット又は JEDICOS 文字コード（JISE）を含む。
10	01-00	なし。
11 ※	01-00 ～ 01-02	バージョン 10 から追加された文字コードを含む。
16	02-00 ～	なし。
20	03-00 ～	なし。
21	03-01 ～	Sea-NACCS フォーマットを含む。

（凡例）

―：該当する Interschema のバージョンはありません。

注※

このバージョンのファイルは，次に示すバージョンの Interschema では使用できません。

- Cosminexus Interschema 01-00（Windows 版）
- Cosminexus Interschema - Definer 01-00（Windows 版）
- Cosminexus Interschema 01-00（HP-UX 版）
- Cosminexus Interschema 01-01（HP-UX 版）

付録 F.2　トランスレータのサポート範囲

トランスレータのサポート範囲について説明します。Interschema では，CII や EDIFACT などの規格は，トランスレータのサポート範囲を超えない範囲でサポートされています。

表 F-2　トランスレータのサポート範囲

項目	サポート範囲
項目長	最大 2,000,000,000 バイト
数値桁数	最大 30 桁（小数部は最大 29 桁）
繰り返し数	最大 2,000,000,000
ネスト数	最大 2,000,000,000
項目数	最大 2,000,000,000
データサイズ	最大 4,000,000,000

付録 G　CII シンタックスルールのサポート範囲

Interschema の CII シンタックスルールのサポート範囲について説明します。

付録 G.1　対応するバージョン

Interschema は，CII シンタックスルールのバージョンは 1.51，2.10，及び 3.00 に対応しています。CII2.10 又は CII3.00 でデータを変換する場合は，CII3 フォーマットを使用します。CII1.51 でデータを変換する場合は，CII3 フォーマット又は従来の CII フォーマットのうちどちらかを使用します。

バージョン 1.10 又は 1.11 で作成された CII 標準形式のデータは，そのまま入力データとして使用できます。バージョン 1.10 又は 1.11 のデータとして出力する場合には，次の対応が必要です。

- CII 1.51 で追加された構造（バイナリデータ）は使用しない。
- シンタックスのバージョンを表す要素（メッセージグループ・ヘッダの「シンタックス ID バージョン」）に，対応させたいバージョンの値をマッピングした MDL ファイルで変換する。

付録 G.2　サポート機能基準との比較

CII サポート機能基準と Interschema のサポート範囲の比較を次に示します。

表 G-1　CII サポート機能基準との比較

基準機能項目			Interschema でのサポート範囲
項目属性	X※1		混在文字列型属性で対応
	B		バイト列型などで対応
	K		2 バイト文字列型属性で対応
	9		暗黙的小数点付数値型又は整数型属性で対応
	N		実数型属性で対応
	Y		日付型属性で対応
項目長	X	2,000 ～ 32,767 文字	最大 32,767 バイト 数値型属性は最大 30 桁（小数部は最大 29 桁）
	B	～ 32,767 バイト	
	K	1,000 ～ 32,766 文字	
	N	18 ～ 30 桁	
	9	18 ～ 30 桁	
	Y	6 桁，8 桁	
標準メッセージ	32,750 バイト～		無制限

基準機能項目		Interschema でのサポート範囲
	ネスト 3 レベル以上	無制限
	繰り返し数 1,000 回以上	最大 999,999,999
	最大項目数 200 項目以上	無制限
オプション	通常／分割モードの選択	対応（メッセージグループ・ヘッダで指定されたモードに従う）
	縮小／拡張モードの選択	対応（メッセージグループ・ヘッダで指定されたモードに従う）
	非透過モード	非サポート
	TYPE-E	非サポート
	バイナリデータの扱い	対応
	短縮メッセージグループ※ 2	入力フォーマットだけ対応
運用メッセージの出力	エラー情報メッセージ	対応
	受信確認メッセージ	対応
	0 件メッセージ	対応
	同報ヘッダ	対応

注※ 1

CII フォーマットでは，１バイト文字列型属性で対応しています。

注※ 2

CII フォーマットでは，対応していません。

トランスレータが扱えるデータのサイズは，最大 4,000,000,000 バイトです。表 G-1 のサポート範囲は，トータルデータサイズが最大値を超えない範囲でサポートされます。

付録 G.3　エラーコードの対応

CII が定めるエラーコードを次に示します。トランスレータの実行時オプションで，エラー情報メッセージ又は受信確認メッセージの出力を指定した場合，トランスレータが出力するエラーメッセージに対応したエラーコードを設定して，CII 運用メッセージを出力します。

表 G-2　エラーコードの対応

エラーコード	エラー内容	対応するトランスレータのエラーメッセージ番号※
（スペース）	エラーなし	－
01	取り決め以外の情報区分コード	－
02	メッセージグループ・ヘッダが見付からない	42

エラーコード	エラー内容	対応するトランスレータのエラーメッセージ番号※
03	メッセージグループ・トレーラが見付からない	43
04	シンタックス ID の不正	38
05	分割識別子シーケンスエラー	40
10	未定義制御タグの検出	47
11	不正データタグの検出	—
12	マルチ明細ヘッダが実行形式 SM テーブル（変換テーブル）上にない	—
13	マルチ明細トレーラが実行形式 SM テーブル（変換テーブル）上にない	—
14	ローカル側（標準側）繰り返しが標準側（ローカル側）繰り返しを超えた	—
15	データのレングスが最大値を超えた	50
16	チェックサム項目の値が数値ではない	49
17	数値変換の項目の値が数値ではない	—
18	標準側データ長がローカル側データ長より大きい	—
19	レコード区分が D ではない（メッセージが見付からない）	44
20	過大レコード長	—
21	メッセージトレーラ（X'FE'）がない	—
22	負のデータあり（9 属性の時など）	—
30	シーケンス No. が昇順ではない	41
31	チェックサムの数値がメッセージグループ・トレーラ上の数値と一致しない	11
32	実行形式 SM テーブルサーチ不能（対応する変換テーブルなし）	—
33	不正文字コードの検出	7，34
34	非透過モード時のメッセージ長不正	—
35	縮小モード中のマルチ明細のネスト，又は拡張モードマルチ明細ヘッダあり	46
36	不正日付の検出	—
40	UNA セグメントの不正	—
41	UNB セグメントなし	—
42	UNH セグメントなし	—
43	UNT セグメントなし	—
44	UNZ セグメントなし	—

エラーコード	エラー内容	対応するトランスレータの エラーメッセージ番号※
81	交換エラー	22 ～ 33，35，36，39，45，48，50
82	同報エラー	51
99	その他のエラー	14 ～ 20，52

（凡例）

－：該当しません。

注※

対応するトランスレータのエラーメッセージ番号とは，プレフィクス，出現元プログラム，メッセージレベルを除いた下 4 桁の数値です。例えば，メッセージ番号「42」はメッセージ ID「KBET0042T-E」に，メッセージ番号「11」はメッセージ ID「KBET0011T-W」に該当します。トランスレータのエラーメッセージについては，「付録 B　トランスレータのメッセージ」を参照してください。

付録 G.4　トランスレータの対応

CII 標準形式データの変換は，次のどれかを使用することを前提とします。

- 電子データ交換ツールの辞書
- Interschema の辞書
- Windows 版のトランスレータで提供する CII3 フォーマット

(1) データ入力時のチェック

メッセージグループ・ヘッダの次の値をチェックします。

- 非透過モード
  0x4D（'M'）の場合は未対応モードのためエラーとします
- 拡張モード
  0x53（'S'）又は 0x20（' '）の場合で，メッセージ中に拡張モード指示子が現れた場合はエラーとします

このほかに，メッセージデータに対して次の場合をエラーとします。

- 縮小モードでマルチ明細のネストがある
- 未定義／非対応データタグが出現した

(2) データ出力時の値

メッセージデータ／バイナリデータを分割モード又は通常モードで出力するかは，メッセージグループ・ヘッダの分割モード要素に従います。CII3 フォーマットの場合，出力データの形式はメッセージグループ・ヘッダの CII シンタックスバージョン要素の値に従います。また，次の要素は変換したデータの内容に従って，マップ式の定義とは別に

設定し直します。

- 同報ヘッダ
  継続区分。
- メッセージグループ・ヘッダ
  フォーマット ID，文字コード（8bit ／ 16bit），非透過モード。
- バイナリデータ・トレーラ
  最終ユニット有効長，全ユニット数。
- メッセージグループ・トレーラ
  最終シーケンス番号，ハッシュトータル 1，2。

メッセージグループ・ヘッダ，トレーラで使用できない文字がある場合は，フォーマットの不正文字コード置換が指定されていてもエラーとします。また，縮小モードでマルチ明細がネストする場合もエラーとします。

(3) ハッシュトータルチェック

CII フォーマットでは，メッセージグループ・ヘッダでトータル項目の指定がある場合，変換データのハッシュトータルチェックを実施します。通常は CII 標準の計算方式（属性の定義に関係なく小数部なしで計算）に従いますが，次の場合では，EIAJ 標準の計算方式（小数部を意識し，メッセージグループ・トレーラの値は小数部 3 桁で出力）にも従います。

- CII 標準形式データをローカル形式データに変換する場合（入力時）
  データが EIAJ 標準に従うと見なす場合，EIAJ 標準及び CII 標準の計算方式で値をチェックします。EIAJ 標準，CII 標準の計算方式どちらにも一致しない場合にワーニングエラーとします。
- ローカル形式データを CII 標準形式データに変換する場合（出力時）
  実行時オプション「-EIAJHASH」を指定し，かつデータが EIAJ 標準に従うと見なす場合，EIAJ 標準の計算方式で値を出力します。このほかの場合では CII 標準の計算方式で出力します。
- データが EIAJ 標準に従うと見なす条件
  データが EIAJ 標準に従うと見なす条件は，メッセージグループ・ヘッダの BPID 版の値が「1x」（x は任意）であることです。

なお，トータル項目に指定されている TFD 項目の要素が数値と見なせない場合，その要素を加算の対象から外します。

CII フォーマットで，EIAJ 標準の計算方式で値をチェックするには，EIAJ 標準メッセージで使用される数値属性項目の TFD データタグ番号と小数部桁数を，EIAJ ハッシュ用ファイル「ethash.ini」に設定しておく必要があります。設定方法については，「5.3(3)　EIAJ ハッシュ用ファイル「ethash.ini」の設定」を参照してください。なお，CII3 フォーマットでは，ethash.ini の定義は不要です。

(4) TFD 項目の省略

CII フォーマットでは，実行時オプション「-CIIT」を指定して CII 標準データを出力する場合，省略の対象となる TFD 項目は出力しません。要素（データエレメント）の内容が次の条件にあてはまる TFD 項目が省略の対象となります。

- X 属性／ K 属性で，すべてスペース。
- 9 属性／ N 属性／ Y 属性で，数値として「0」（± 0.0 も 0 と見なす）。
- B 属性で，各バイトがすべて「0x00」。

省略の対象となる TFD 項目の属性とデータタグ番号の対応を，CII データタグファイル「etciitag.ini」に設定しておく必要があります。設定方法については，「5.3(2)　CII データタグファイル「etciitag.ini」の設定」を参照してください。なお，CII3 フォーマットでは，省略できる TFD 項目は常に出力されません。また，etciitag.ini の定義は不要です。

また，マルチ明細（ネストしたマルチ明細も含めて）中のすべての TFD 項目が省略された場合，マルチ明細そのものの省略は次のとおりです。

- 縮小モードの場合
  マルチ明細ヘッダ，マルチ明細トレーラだけ出力します。
- 拡張モードの場合
  マルチ明細全体を省略します。

(5) バイナリデータ

CII フォーマットでは，バイナリデータ（ユニット部分）を一つの要素として，その前にバイナリデータ長要素（バイナリデータの長さを表す要素）を設定してあります。CII 標準形式データを入力する場合はバイナリデータ長要素を自動生成します。CII 標準形式データを出力する場合，バイナリデータ長要素に定義された長さ分のデータを，CII 標準の形式に変換して出力します。

(6) テストデータ

入力データがテスト用のデータだった場合，テスト用のデータであることを示すメッセージ（インフォメーション）を出力します。メッセージグループ・ヘッダの運用モードが「0x31」の場合は，テスト用のデータです。

(7) 運用メッセージ

実行時オプション「-CIIR」又は「-CIIE」を指定して出力された，受信確認メッセージ又はエラー情報メッセージは，入力する 1 メッセージグループにつき 1 メッセージデータを出力します。複数のメッセージグループデータを入力した場合，入力データ中の連続するメッセージグループで，「発信 VAN コード」「発信センタコード」「発信者コード」，「受信 VAN」「受信センタコード」「受信者コード」，「BPID 機関」「BPID サブ機関」「BPID 版」が同じメッセージグループに対応するデータを，一つのメッセージグ

ループとして出力します。

入力するデータにエラーがあり，メッセージグループ・トレーラまでを解釈できない場合は，該当するメッセージグループ・トレーラの部分をスペースにして運用メッセージを出力します。

付録 G.5　EIAJ 標準メッセージの対応範囲

EIAJ 標準メッセージのデータとして変換できる制限範囲を次に示します。辞書をベースとしてデータを変換する場合には，EIAJ 1D 版の辞書を使用してください。

なお，トランスレータが変換の対象データを EIAJ 標準に従うと見なす条件は，メッセージグループ・ヘッダの BPID 版の値が「1x」（x は任意）であることです。

- 分割／縮小モードを使用する。
- N 属性，K 属性項目を使用しない。
- 240 以上のタグ番号（項目 No）を使用しない。
- 240 バイト以上の長さを持つデータ項目を使用しない。
- 明細番号付マルチ明細，マルチ明細のネストを使用しない。
- EIAJ 標準形式の出力データを変換する場合に，ハッシュトータルチェックを実施するときは，実行時オプション「-EIAJHASH」を指定する（又は小数部 3 桁の 9 属性項目を指定する）。
- 文字セットは標準のものを使用する（8bit JIS）。
- 最大メッセージ長を 32,767 バイト以下にする。

付録 H　UN/EDIFACT のサポート範囲

Interschema の UN/EDIFACT のサポート範囲について説明します。

付録 H.1　対応するバージョン

Interschema が対応する UN/EDIFACT のバージョンは，Version.4 の中の Part1，Part2，Part3，Part8 です。

（1）EDIFACT フォーマットで Version.1 ～ 3 のデータを変換する場合

Version.1 ～ 3 で作成された UN/EDIFACT のデータを入力データとして使用する場合，セグメントの明示的なネスト表現部分は無条件に読み飛ばします。Version.1 ～ 3 のデータとして出力する場合には，次の対応が必要です。

- Version.4 で追加された構造（INTERACTIVE，パッケージなど）は使用しないでください。
  なお，サービスセグメント関連の Version.4 と Version.1 ～ 3 の間で，次に示す相違点があります。

表 H-1　サービスセグメント関連の相違点（Version.4 と Version.1 ～ 3）

コンポーネント	Version.4	Version.1 ～ 3
UNG	n0038，S006，S007，n0051，S008 は選択（Conditional）です。	n0038，S006，S007，n0051，S008 は必須（Mandatory）です。
UNH	S016，S017，S018 が追加されました。	－
S001	n0080，n0133 が追加されました。	－
S002	n0042 が追加されました。	－
S003	n0046 が追加されました。	－
S009	n0010，n0113 が追加されました。	－
n0008, n0014	英数字で 35 桁まで指定できます。	英数字で 14 桁まで指定できます。
n0017	数値 8 桁（CCYYMMDD）で指定します。	数値 6 桁（YYMMDD）で指定します。
n0051	英数字で 3 桁まで指定できます。	英数字で 2 桁まで指定できます。
n0074	10 桁まで指定できます。	6 桁まで指定できます。

（凡例）
　－：該当しません。

注
　n 付きのコンポーネントは要素を表します。

● UNA セグメントなしで UNOB 文字セットのデータを出力する場合，FDL エディタで EDIFACT フォーマットを次のように修正した FDL ファイルを使用した MDL ファイルで変換します。修正する内容を次に示します。
- INTERCHANGE 構造から子コンポーネント UNA を削除する。
- ROOT 構造のセパレータ指定で，解放文字の指定を削除する。
- セパレータ定義で，値を UNOB 文字セットの場合の初期値へ変更する。UNOB 文字セットの場合の初期値を次に示す。

セパレータ名	値（バイト列）
COMPONENT	0x1F
ELEMENT	0x1D
SEGMENT	0x1C

● シンタックスのバージョンを表す要素（0002 Syntax Version Number 要素）に，対応させたいバージョンの値をマッピングした MDL ファイルで変換します。

（2）EDIFACT4 フォーマットで Version.1 ～ 3 のデータを変換する場合

Version.1 ～ 3 で作成された UN/EDIFACT のデータを入力データとして使用する場合の対応は，EDIFACT フォーマットの場合と同じです。「(1)　EDIFACT フォーマットで Version.1 ～ 3 のデータを変換する場合」を参照してください。Version.1 ～ 3 の UN/EDIFACT のデータを出力する場合は，Version.1 ～ 3 用の EDIFACT4 フォーマットを使用して MDL ファイルを作成してください。

付録 H.2　サポート機能基準との比較

UN/EDIFACT サポート機能基準と Interschema のサポート範囲の比較を次に示します。

表 H-2　UN/EDIFACT サポート機能基準との比較

基準機能項目		Interschema でのサポート範囲
サイズ	項目数	無制限
	項目長	文字列型属性は最大 999,999,999 バイト 数値型属性は最大 30 桁（小数部は最大 29 桁）
	繰り返し数	最大 999,999,999
	ネスト数	無制限
	メッセージ長	無制限
データ内容	文字コード	次の文字セットに対応 • A レベル文字セット（英大文字・数字）（UNOA） • B レベル文字セット（7 ビット文字）（UNOB） • 拡張手法による日本語文字コード（ISO-2022-JP）（UNOX）
構造	バッチ	対応

基準機能項目		Interschema でのサポート範囲
	対話型	対応
	関連データ（パッケージ）	対応

トランスレータが扱えるデータのサイズは，最大 4,000,000,000 バイトです。表 H-2 のサポート範囲は，トータルデータサイズが最大値を超えない範囲でサポートされます。

付録 H.3　エラー情報

トランスレータが独自に出力する受信確認，エラー情報メッセージはありません。変換時のエラー情報はログファイルとして出力します。ログファイルの参照方法については，「付録 A　トラブルシューティング」を参照してください。

付録 H.4　トランスレータの対応

UN/EDIFACT 形式データの変換は，次のどれかを使用することを前提とします。

- 電子データ交換ツールの辞書
- Interschema の辞書
- Windows 版のトランスレータで提供する EDIFACT4 フォーマット

属性型など FDL ／ MDL で定義される内容以外でのトランスレータの対応は次のとおりです。

(1) テストデータ

入力データがテスト用のデータだった場合，テスト用のデータであることを示すメッセージ（インフォメーション）を出力します。テストフラグが格納されている要素（0035 Test Indicator）の値が「1」から「4」までの場合は，テスト用のデータです。

付録 I　JEDICOS のサポート範囲

Interschema の JEDICOS のサポート範囲について説明します。

付録 I.1　対応するバージョン

Interschema が対応する JEDICOS のバージョンは，Version.2 です。

付録 I.2　サポート機能基準との比較

UN/EDIFACT と同じです。「付録 H.2　サポート機能基準との比較」の「表 H-2　UN/EDIFACT サポート機能基準との比較」を参照してください。ただし，文字コードには UN/EDIFACT の UNOX を使用しています。

付録 J　Sea-NACCS のサポート範囲

Interschema の Sea-NACCS のサポート範囲について説明します。

付録 J.1　対応するバージョン

Interschema では，次の表に示す規定内容に従います。

表 J-1　Interschema で従う規定内容

規定内容	Sea-NACCS フォーマット
1999 年 6 月現在の規定内容（1999 年 10 月運用開始）	• Sea-NACCS EDI 電文フォーマット（旧） • Sea-NACCS EDIFACT 電文フォーマット（旧）
2008 年 6 月現在の規定内容（2008 年 10 月システム更改）	• Sea-NACCS EDI 電文フォーマット • Sea-NACCS EDIFACT 電文フォーマット

なお，Sea-NACCS EDIFACT 電文でのシンタックスルールは Version.3，文字セットは UNOA です（ただし，Sea-NACCS での規定を優先します）。

付録 J.2　サポート機能基準との比較

Sea-NACCS EDI 電文のサポート機能基準と Interschema でのサポート範囲の比較を次に示します。

表 J-2　Sea-NACCS EDI 電文サポート機能基準との比較

基準機能項目		Interschema でのサポート範囲
属性	an 型	1 バイト文字列型で対応（文字セットは JIS7A）
	n 型	整数又は実数型で対応（文字セットは JIS7A） ただし，30 桁を超える項目は文字列型で対応 日時を表す場合は，日時型（日付，時刻，日付時刻）でも対応（推奨は整数型）
	j 型	混在文字列型で対応（文字セットは EUC）
文字コード	－	JIS7A 又は EUC で対応
その他	電文分割※ 1	未サポート
	CSV 形式※ 2	未サポート
	XML 電文※ 3	未サポート

（凡例）

－：該当しません。

注※ 1

Sea-NACCS フォーマット（旧）の場合，通関情報処理センターのセンターホストから出力される処理結果電文が複数に分割されているときは，トランスレータで変換（入力）する前に電文を組み立てる必要があります。

注※ 2

管理資料が該当します。

注※ 3

システムとしては対応しません。

Sea-NACCS EDI 電文長は最大 500,000 バイトです。Interschema では，Sea-NACCS EDI 電文長についてサイズの制限はありません。

付録 J.3　エラー情報

トランスレータが独自に出力する受信確認，エラー情報メッセージはありません。通関情報処理センターのセンターホストから返ってくる受信確認，エラー情報メッセージは，Sea-NACCS EDI 電文又は Sea-NACCS EDIFACT 電文の 1 メッセージとして変換できます（トランスレータにとっては，種別は入力になります）。

付録 J.4　トランスレータの対応

変換処理は，前提として使用するものがフォーマットによって次のように異なります。

- Sea-NACCS EDI 電文フォーマット（旧），及び Sea-NACCS EDIFACT 電文フォーマット（旧）の場合
  電子データ交換ツールの海上貨物通関用辞書を使用することを前提とします。
- Sea-NACCS EDI 電文フォーマット，及び Sea-NACCS EDIFACT 電文フォーマットの場合
  Interschema がサンプルとして提供するフォーマットを使用することを前提とします。

属性型など FDL ／ MDL で定義される内容以外でのトランスレータの対応は次のとおりです。

（1）Sea-NACCS EDI 電文を入力する場合（処理結果通知電文，出力情報電文）

Sea-NACCS EDI 電文を入力データとして変換する場合，FDL ／ MDL ファイルで定義されているすべてのコンポーネントのデータが現れる前に入力データが終了したときは，入力データ終了以降に相当するコンポーネントのデータはスペースが入力されたものとして扱います。この場合，数値属性の場合は値「0」が入力されたことになります。

（2）Sea-NACCS EDI 電文を出力する場合（処理要求電文）

Sea-NACCS EDI 電文を出力データとして変換する場合，電文末尾のスペース項目をカットして，電文長も補正します。このほかの場合，次のように変換します。

- マップ式が評価できない場合（入力データの出現回数が少ない場合など）は，デフォルト式の値を出力します。
- NULL マップ式が定義された要素に対しては，スペースを出力します。
- 実行時オプション「-SNCODE」が指定されている場合，文字コードチェックを厳密に実施します（これは Sea-NACCS で規定する文字コードと，Interschema で扱う文字コードの範囲が異なるためです）。

（3） Sea-NACCS EDIFACT 電文を変換する場合

Sea-NACCS EDIFACT 電文を変換する場合，FDL ／ MDL ファイル内で定義されている以外には，特別な変換処理はされません。通常の UN/EDIFACT 形式データとは次の点が異なります。

- テストモードの判定がありません（Sea-NACCS ではテストフラグを使用しないため，通常のデータとテスト用のデータを区別できません）。
- 処理要求電文（トランスレータでは出力）の場合，実行時オプション「-SNCODE」が指定されているときに，文字コードチェックを厳密に実施します。

付録 K　文字コード

トランスレータが扱う文字コード，及び文字種別について説明します。また，文字コード間での変換，及び EBCDIC ／ EBCDIK のコードについても説明します。

付録 K.1　文字コード一覧

トランスレータが扱う文字コードの一覧（詳細）を次に示します。

表 K-1　文字コード一覧（詳細）

コード指定子	コード範囲 0x < 文字種別 > ※ 1	種別	説明
JIS8	<0201R> 00-7f <0201K> a1-df	1 バイト	JIS-X0201 (8bit) JIS-X0201 をフルサポートしています。
JIS7	<ISO> 00-7f	1 バイト	ISO/IEC 646 (7bit) 一般的には ASCII コードです。 JIS-X0201 のラテン文字部分として扱います。
JIS7A	<ISO> 20-60,7b-7e	1 バイト	ISO/IEC 646 制御コードと英小文字を除く UN/EDIFACT A セット文字用コードです。
ISO8859	<ISO> 00-7f <8859G1> a1-ff	1 バイト	ISO-8859 GL は JIS7 での扱いです。 GR は他コードへの変換できません。
EBCDIC	—	1 バイト	日立 EBCDIC JIS-X0201 との間でテーブル変換します。
EBCDIK	—	1 バイト	日立 EBCDIK JIS-X0201 との間でテーブル変換します。
IBM_EBCDIC	—	1 バイト	IBM EBCDIC EBCDIC-C-US (cp00281) です。 JIS-X0201 との間でテーブル変換します。 かな文字は含みません。
IBM_EBCDIK	—	1 バイト	IBM EBCDIK EBCDIC-JP-Kana (cp00290 拡張前) です。 JIS-X0201 との間でテーブル変換します。 英小文字は含みません。
JISK	<0208>[1st] 21-7e <0208>[2nd] 21-7e	2 バイト	JIS-X0208 (JIS 漢字) JIS-X0208-1990 です。 文字チェックは Unicode テーブルとのマッチングによって行います。

コード指定子	コード範囲 0x < 文字種別 > ※ 1	種別	説明
JISKS	JISK を参照	2 バイト	JIS-X0208 シフト IN/OUT 付き JISK に対してシフトコードを付けたものです。 開始：0x1B2442（2 バイト文字へ遷移） 終了：0x1B284A（1 バイト文字へ遷移） 終了シフトコード以降は，JIS-X0201 のスペース文字 (0x20) だけ許します。ただし，データとしては扱いません。
KEIS	(基本)[1st] a1-cf (基本)[2nd] a1-fe (拡張)[1st] d0-fe (拡張)[2nd] a1-fe (ユーザ)[1st] 41-a0 (ユーザ)[2nd] a1-fe	2 バイト	KEIS (日立漢字コード) KEIS ’ 83 （KEIS ’ 90 で追加された文字にも対応）です。 JIS-X0208 との間で変換できます。
KEISS	KEIS を参照	2 バイト	KEIS シフト IN/OUT 付き KEIS に対してシフトコードを付けたものです。 開始：0x0A42（2 バイト文字へ遷移） 終了：0x0A41（1 バイト文字へ遷移） 終了シフトコード以降は，EBCDIK のスペース文字 (0x40) だけ許します。ただし，データとしては扱いません。
SJISK	[1st] 81-9f, e0-ef [2nd] 40-7e,80-fc (IBM 拡張)[1st] fa-fc (IBM 拡張)[2nd]40-7e,80-fc (ユーザ)[1st] f0-f9 (ユーザ)[2nd] 40-7e,80-fc	2 バイト	Shift-JIS 2 バイトコード SJIS の 2 バイトコードだけで構成されます。SJIS を参照してください。
IBM	[1st] 41-68 [2nd] 41-fe (ユーザ)[1st] 69-7f (ユーザ)[2nd] 41-fe	2 バイト	IBM DBCS-Host (IBM 漢字コード) IBM'83 です。 SJIS との間でテーブル変換します。
JEF	[1st] a1-fe ※ 2 [2nd] a1-fe ※ 2 (拡張)[1st] 41-7d, 7f (拡張)[2nd] a1-fe (ユーザ)[1st] 80-a0 (ユーザ)[2nd] a1-fe	2 バイト	JEF (富士通漢字コード) JIS-C6226-1978 との間で変換できます。 JEF 拡張領域，ユーザ定義文字を共に外字領域として扱います。
CJIS	<0208>[1st] 21-7e <0208>[2nd] 21-7e <0212>[1st] a1-fe <0212>[2nd] 21-7e	2 バイト	CII 用 JIS-X0208 + JIS-X0212 CII 2.10 以降で使用される組み合わせコードです。 JIS-X0212 に対して MSB 操作をします。

コード指定子	コード範囲 0x < 文字種別 > ※ 1	種別	説明
ISOJP	<ISO/0201R> 00-7f <0208>[1st] 21-7e <0208>[2nd] 21-7e	混在	ISO-2022-Jp ISO-2022 に従って 7bit の範囲で ASCII と JIS-X0208 を切り替える文字コードです。JIS-X0201(かな), JIS-X0212 は使用できません。 シフトコードは次のとおりです。 • 0x1B284A：JIS-X0201 (ROMAN) • 0x1B2440：JIS-C6226 (JIS-X0208-1978) • 0x1B2442：JIS-X0208 Interschema の文字コード変換で出力する場合，次のシフトコードは出力されません。 • 0x1B2842：ISO/IEC 646
SJIS	<0201R> 00-7f <0201K> a1-df [1st] 81-9f, e0-ef [2nd] 40-7e,80-fc (ユーザ)[1st] f0-f9 (ユーザ)[2nd] 40-7e,80-fc (IBM 拡張)[1st] fa-fc (IBM 拡張)[2nd]40-7e,80-fc	混在	Shift-JIS 1 バイトコード：JIS-X0201 2 バイトコード：JIS-X0208 を変換したものです。 IBM 拡張文字は，Unicode との間で変換できます。
EUC	<0201R> 00-7f <0201K>(SC) 8e <0201K> a1-df <0208>[1st] a1-fe <0208>[2nd] a1-fe <0212>(SC) 8f <0212>[1st] a1-fe <0212>[2nd] a1-fe (ユーザ A)[1st] a1-fe (ユーザ A)[2nd] 21-7e (ユーザ B)[1st] f5-fe (ユーザ B)[2nd] a1-fe	混在	Extended UNIX Code JIS-X0201(かな) と JIS-X0212 に対してはシフトコード（1 バイト，その文字だけに有効）を付けます。 ユーザ定義文字は A（EUC-HJ の外字領域），B どちらでも使用できます。
UNIC	(領域 A)[1st] 00-4d (領域 A)[2nd] 00-ff (領域 I)[1st] 4e-9f (領域 I)[2nd] 00-ff (領域 R)(ユーザ)[1st] e0-f8 (領域 R)(ユーザ)[2nd] 00-ff	混在	Unicode (UCS-2) 各文字を 2 バイトで表します。 JIS コードとの変換は対応テーブルによります。
UCS4	UNIC を参照	混在	Unicode (UCS-4) 各文字を 4 バイトで表します。変換テーブルは UNIC と同様です。

コード指定子	コード範囲 0x < 文字種別 > ※ 1	種別	説明
UTF8	UNIC を参照	混在	Unicode (UTF-8) 各文字を UCS の文字コードで実際に使用されているビット数に応じた長さで表します。 変換テーブルは UNIC と同様です。
UTF16	UNIC を参照	混在	Unicode (UTF-16) U+10000 から U+10FFFF の範囲を Unicode 規格内で予約された領域を使い，二つの 16 ビットデータの組み合わせで 1 文字を表現（サロゲートペア）します。 サロゲートペア以外は UCS-2 と同様です。 変換テーブルは UNIC と同様です。
JISE	<0201R> 00-7f <0201K> a1-df <0208>[1st] 21-7e <0208>[2nd] 21-7e <0212>[1st] a1-fe <0212>[2nd] a1-fe	混在	JEDICOS 文字セット ISO-2022 に従って JIS-X0201，JIS-X0208，JIS-X0212 を切り替えます。 シフトコードは次のとおりです。 • 0x1B284A：JIS-X0201 (ROMAN) • 0x1B2949：JIS-X0201 (かな) • 0x1B2442：JIS-X0208 • 0x1B242944：JIS-X0212 Interschema の文字コード変換で出力する場合，次のシフトコードは出力されません。 • 0x1B2842：ISO/IEC 646 • 0x1B2849：JIS-X0201 (かな) • 0x1B2440：JIS-C6226 (JIS-X0208-1978) • 0x1B242844：JIS-X0212
KEIS_EBCDIC_MIX	KEIS，EBCDIC を参照	混在	KEIS+EBCDIC 混在 Shift-In：0x0A41（1 バイト文字へ遷移） Shift-Out：0x0A42（2 バイト文字へ遷移）
KEIS_EBCDIK_MIX	KEIS，EBCDIK を参照	混在	KEIS+EBCDIK 混在 Shift-In：0x0A41（1 バイト文字へ遷移） Shift-Out：0x0A42（2 バイト文字へ遷移）
IBM_EBCDIC_MIX	IBM，IBM_EBCDIC を参照	混在	IBM+IBM_EBCDIC 混在 Shift-In：0x0F（1 バイト文字へ遷移） Shift-Out：0x0E（2 バイト文字へ遷移）
IBM_EBCDIK_MIX	IBM，IBM_EBCDIK を参照	混在	IBM+IBM_EBCDIK 混在 Shift-In：0x0F（1 バイト文字へ遷移） Shift-Out：0x0E（2 バイト文字へ遷移）
JEF_EBCDIC_MIX	JEF，IBM_EBCDIC を参照	混在	JEF+IBM_EBCDIC 混在 Shift-In：0x29（1 バイト文字へ遷移） Shift-Out：0x28（2 バイト文字へ遷移）
JEF_EBCDIK_MIX	JEF，IBM_EBCDIK を参照	混在	JEF+IBM_EBCDIK 混在 Shift-In：0x29（1 バイト文字へ遷移） Shift-Out：0x28（2 バイト文字へ遷移）

（凡例）

1st：第 1 バイトです。

2nd：第 2 バイトです。

ユーザ：ユーザ定義文字です。

SC：シフトコードです。

1 バイト：1 バイト文字コードです。

2 バイト：2 バイト文字コードです。

混在：1 バイト／ 2 バイト混在文字コードです。

－：該当しません。

注※ 1

文字種別については，「付録 K.2　文字種別」を参照してください。

注※ 2

2 バイト文字の値として「0xa1a1」を除きます。

付録 K.2　文字種別

トランスレータが扱う文字コードは JIS コードを基本とし，それぞれの文字を次に示す文字種別に分類して操作します。

表 K-2　文字種別一覧

文字種別	意味	説明
ISO	ISO/IEC 646	JIS-X0201 (ROMAN) と同様に扱います。 ISO/IEC 646 の文字を JIS-X0201 (ROMAN) にそのまま代入できます。
0201R	JIS-X0201(ROMAN)	7bit JIS です。ISO/IEC 646 と同様に扱います。 0x00 ～ 0x1f：制御コード 0x20 ～ 0x7e：印字可能文字 0x7f：DEL
0201K	JIS-X0201(かな)	JIS-X0201 のかな文字部分：0xa1 ～ 0xdf
0201K_G1	JIS-X0201(かな) G1	JIS-X0201(かな) と同様です。 解放文字処理などで文字種別（シフトコード）を区別するために使用します。
8859G1	ISO-8859(G1)	ISO-8859 の G1(GR) 文字部分：0xa1 ～ 0xff
C6226	JIS-C6226	JIS-X0208-1978 です。 各バイトは 0x21 ～ 0x7e の範囲となります。 JIS-X0208 へ変換できます。 JIS-C6226 から JIS-X0208-1983 の変更は次のとおりです。 • 削除文字なし • 追加文字あり • コード入れ替え 22 文字 • 字形変更されたものあり

文字種別	意味	説明
0208	JIS-X0208	JIS-X0208-1990 です。 各バイトは 0x21 ～ 0x7e の範囲となります。 トランスレータでは JIS-X0208-1983 とは区別しません。
0212	JIS-X0212	JIS-X0212-1990 です。 各バイトは 0x21 ～ 0x7e の範囲となります。 JIS-X0208 との変換はできません。
0212_G1	JIS-X0212 G1	JIS-X0212 と同様です。 解放文字処理などで文字種別（シフトコード）を区別するために使用します。
UTF	Unicode(サロゲートペア)	サロゲートペアで表される文字を UCS-4 表現したものです。 Unicode 内でだけ扱えます（UCS-2 を除きます)。 UTF-16 での 1st 値：0xd800 ～ 0xdbff UTF-16 での 2nd 値：0xdc00 ～ 0xdfff
KEIS_USER	KEIS ユーザ定義文字	1st 値：0x41 ～ 0xa0 2nd 値：0xa1 ～ 0xfe
IBM_USER	IBM ユーザ定義文字	1st 値：0x69 ～ 0x7f 2nd 値：0x41 ～ 0xfe
JEF_USER	JEF ユーザ定義文字	1st 値：0x41 ～ 0x7d, 7f, 0x80 ～ 0xa0 2nd 値：0xa1 ～ 0xfe
UCS2_USER	UCS2 ユーザ定義文字	Unicode のユーザ定義文字は，UCS-2 の範囲だけ対応します。 1st 値：0xe0 ～ 0xf8 2nd 値：0x00 ～ 0xff
SJIS_USER	SJIS ユーザ定義文字	1st 値 0xfa ～ 0xfc は IBM 拡張漢字で，Unicode 変換時は通常文字扱いとなります。 1st 値：0xf0 ～ 0xfc 2nd 値：0x40 ～ 0x7e, 0x80 ～ 0xfc
EUC_USER	EUC ユーザ定義文字	[A] 1st 値：0xa1 ～ 0xfe 2nd 値：0x21 ～ 0x7e [B] 1st 値：0xf5 ～ 0xfe 2nd 値：0xa1 ～ 0xfe

（凡例）

1st 値：第 1 バイトの値です。

2nd 値：第 2 バイトの値です。

付録 K.3　文字コード変換

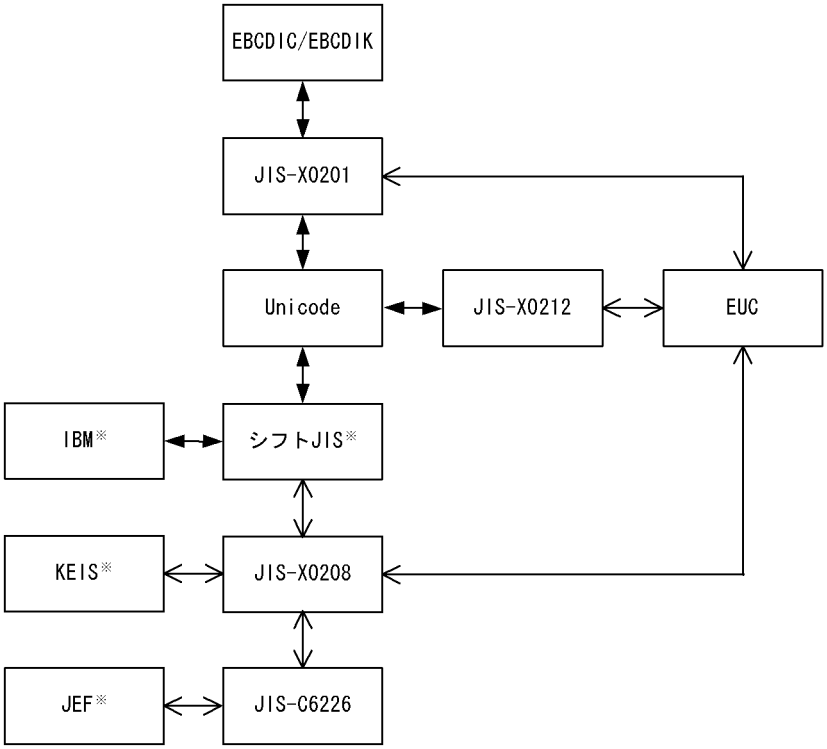
各文字コード間では，1 バイト文字コードから 2 バイト文字コード，及び 2 バイト文字コードから 1 バイト文字コードへの変換はできません。1 バイト文字コードと 2 バイト

文字コード間の変換以外は，対応する文字種別及び値が存在するとき変換できます。ただし，ユーザ定義文字は変換できません。

なお，変換は 1 文字単位で処理されます。

文字コードと文字種別の変換の関連図を次に示します。

図 K-1　文字コードと文字種別の変換の関連



（凡例）
◀▶ ：テーブル変換
⟷ ：アルゴリズム変換

注※　2バイト文字の部分だけ該当します。

文字コードと文字種別の詳細については，「付録 K.1　文字コード一覧」及び「付録 K.2　文字種別」を参照してください。

付録 K.4　EBCDIC ／ EBCDIK のコード表

EBCDIC ／ EBCDIK のコード表を次に示します。

コード表のコードは 16 進表示です。横は上位 4 ビット，縦は下位 4 ビットを表します。各カラムの上段はコードの 16 進値，下段は対応文字（印字できる文字だけ表記）を表し

ます。

(1) EBCDIC コードから JIS コードへの変換

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	10	80	90	20	26	2D	BA	C3	CA	D1	D8	7B	7D	24	30
					(SP)	&	−	ｺ	ﾃ	ﾊ	ﾑ	ﾘ	{	}	$	0
1	01	11	81	91	A0	A9	2F	BB	61	6A	7E	D9	41	4A	9F	31
						ｳ	/	ｻ	a	j	~	ﾙ	A	J		1
2	02	12	82	16	A1	AA	B2	BC	62	6B	73	DA	42	4B	53	32
					。	ｪ	ｲ	ｼ	b	k	s	ﾚ	B	K	S	2
3	03	13	83	93	A2	AB	B3	BD	63	6C	74	DB	43	4C	54	33
					「	ｫ	ｳ	ｽ	c	l	t	ﾛ	C	L	T	3
4	9C	9D	84	94	A3	AC	B4	BE	64	6D	75	DC	44	4D	55	34
					」	ｬ	ｴ	ｾ	d	m	u	ﾜ	D	M	U	4
5	09	0A	85	95	A4	AD	B5	BF	65	6E	76	DD	45	4E	56	35
					、	ｭ	ｵ	ｿ	e	n	v	ﾝ	E	N	V	5
6	86	08	17	96	A5	AE	B6	C0	66	6F	77	DE	46	4F	57	36
					・	ｮ	ｶ	ﾀ	f	o	w	ﾞ	F	O	W	6
7	7F	87	1B	04	A6	AF	B7	C1	67	70	78	DF	47	50	58	37
					ヲ	ｯ	ｷ	ﾁ	g	p	x	ﾟ	G	P	X	7
8	97	18	88	98	A7	B0	B8	C2	68	71	79	E0	48	51	59	38
					ｧ	ｰ	ｸ	ﾂ	h	q	y		H	Q	Y	8
9	8D	19	89	99	A8	B1	B9	60	69	72	7A	E1	49	52	5A	39
					ｨ	ｱ	ｹ	`	i	r	z		I	R	Z	9
A	8E	92	8A	9A	5B	5D	7C	3A	C4	CB	D2	E2	E8	EE	F4	FA
					[	]	|	:	ﾄ	ﾋ	ﾒ					
B	0B	8F	8B	9B	2E	5C	2C	23	C5	CC	D3	E3	E9	EF	F5	FB
					.	¥	,	#	ﾅ	ﾌ	ﾓ					
C	0C	1C	8C	14	3C	2A	25	40	C6	CD	D4	E4	EA	F0	F6	FC
					<	*	%	@	ｺ	ﾍ	ﾔ					
D	0D	1D	05	15	28	29	5F	27	C7	CE	D5	E5	EB	F1	F7	FD
					(	)	_	’	ﾇ	ﾎ	ｭ					
E	0E	1E	06	9E	2B	3B	3E	3D	C8	CF	D6	E6	EC	F2	F8	FE
					+	;	>	=	ﾈ	ﾏ	ﾖ					
F	0F	1F	07	1A	21	5E	3F	22	C9	D0	D7	E7	ED	F3	F9	FF
					!	^	?	″	ﾉ	ﾐ	ﾗ					

（凡例）

(SP)：スペース文字です。

(2) JIS コードから EBCDIC コードへの変換

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	10	40	F0	7C	D7	79	97	20	30	41	58	76	9F	B8	DC
			(SP)	0	@	P	`	p				ー	タ	ミ		
1	01	11	4F	F1	C1	D8	81	98	21	31	42	59	77	A0	B9	DD
			!	1	A	Q	a	q			。	ア	チ	ム		
2	02	12	7F	F2	C2	D9	82	99	22	1A	43	62	78	AA	BA	DE
			"	2	B	R	b	r			「	イ	ツ	メ		
3	03	13	7B	F3	C3	E2	83	A2	23	33	44	63	80	AB	BB	DF
			#	3	C	S	c	s			」	ウ	テ	モ		
4	37	3C	E0	F4	C4	E3	84	A3	24	34	45	64	8A	AC	BC	EA
			$	4	D	T	d	t			、	エ	ト	ヤ		
5	2D	3D	6C	F5	C5	E4	85	A4	25	35	46	65	8B	AD	BD	EB
			%	5	E	U	e	u			・	オ	ナ	ユ		
6	2E	32	50	F6	C6	E5	86	A5	06	36	47	66	8C	AE	BE	EC
			&	6	F	V	f	v			ヲ	カ	ニ	ヨ		
7	2F	26	7D	F7	C7	E6	87	A6	17	08	48	67	8D	AF	BF	ED
			'	7	G	W	g	w			ァ	キ	ヌ	ラ		
8	16	18	4D	F8	C8	E7	88	A7	28	38	49	68	8E	B0	CA	EE
			(	8	H	X	h	x			ィ	ク	ネ	リ		
9	05	19	5D	F9	C9	E8	89	A8	29	39	51	69	8F	B1	CB	EF
			)	9	I	Y	i	y			ゥ	ケ	ノ	ル		
A	15	3F	5C	7A	D1	E9	91	A9	2A	3A	52	70	90	B2	CC	FA
			*	:	J	Z	j	z			ェ	コ	ハ	レ		
B	0B	27	4E	5E	D2	4A	92	C0	2B	3B	53	71	9A	B3	CD	FB
			+	;	K	[	k	{			ォ	サ	ヒ	ロ		
C	0C	1C	6B	4C	D3	5B	93	6A	2C	04	54	72	9B	B4	CE	FC
			,	<	L	¥	l	|			ャ	シ	フ	ワ		
D	0D	1D	60	7E	D4	5A	94	D0	09	14	55	73	9C	B5	CF	FD
			−	=	M	]	m	}			ュ	ス	ヘ	ン		
E	0E	1E	4B	6E	D5	5F	95	A1	0A	3E	56	74	9D	B6	DA	FE
			.	>	N	^	n	~			ョ	セ	ホ	゛		
F	0F	1F	61	6F	D6	6D	96	07	1B	E1	57	75	9E	B7	DB	FF
			/	?	O	_	o				ッ	ソ	マ	゜		

（凡例）

(SP)：スペース文字です。

（3） EBCDIK コードから JIS コードへの変換

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	10	80	90	20	26	2D	6A	73	BF	77	79	7B	7D	24	30
					(SP)	&	−	j	s	ソ	w	y	{	}	$	0
1	01	11	81	91	A1	AA	2F	6B	B1	C0	7E	7A	41	4A	9F	31
					。	ｪ	/	k	ｱ	ﾀ	~	z	A	J		1
2	02	12	82	16	A2	AB	62	6C	B2	C1	CD	E0	42	4B	53	32
					「	ｫ	b	l	ｲ	ﾁ	ﾍ		B	K	S	2
3	03	13	83	93	A3	AC	63	6D	B3	C2	CE	E1	43	4C	54	33
					」	ｬ	c	m	ｳ	ﾂ	ﾎ		C	L	T	3
4	9C	9D	84	94	A4	AD	64	6E	B4	C3	CF	E2	44	4D	55	34
					、	ｭ	d	n	ｴ	ﾃ	ﾏ		D	M	U	4
5	09	0A	85	95	A5	AE	65	6F	B5	C4	D0	E3	45	4E	56	35
					・	ｮ	e	o	ｵ	ﾄ	ﾐ		E	N	V	5
6	86	08	17	96	A6	AF	66	70	B6	C5	D1	E4	46	4F	57	36
					ヲ	ｯ	f	p	ｶ	ﾅ	ﾑ		F	O	W	6
7	7F	87	1B	04	A7	A0	67	71	B7	C6	D2	E5	47	50	58	37
					ｧ		g	q	ｷ	ﾆ	ﾒ		G	P	X	7
8	97	18	88	98	A8	B0	68	72	B8	C7	D3	E6	48	51	59	38
					ｨ	ｰ	h	r	ｸ	ﾇ	ﾓ		H	Q	Y	8
9	8D	19	89	99	A9	61	69	60	B9	C8	D4	E7	49	52	5A	39
					ｩ	a	i	`	ｹ	ﾈ	ﾔ		I	R	Z	9
A	8E	92	8A	9A	5B	5D	7C	3A	BA	C9	D5	DA	E8	EE	F4	FA
					[	]	|	:	ｺ	ﾉ	ｭ	ﾚ				
B	0B	8F	8B	9B	2E	5C	2C	23	74	75	78	DB	E9	EF	F5	FB
					.	¥	,	#	t	u	x	ﾛ				
C	0C	1C	8C	14	3C	2A	25	40	BB	76	D6	DC	EA	F0	F6	FC
					<	*	%	@	ｻ	v	ﾖ	ﾜ				
D	0D	1D	05	15	28	29	5F	27	BC	CA	D7	DD	EB	F1	F7	FD
					(	)	_	'	ｼ	ﾊ	ﾗ	ﾝ				
E	0E	1E	06	9E	2B	3B	3E	3D	BD	CB	D8	DE	EC	F2	F8	FE
					+	;	>	=	ｽ	ﾋ	ﾘ	ﾞ				
F	0F	1F	07	1A	21	5E	3F	22	BE	CC	D9	DF	ED	F3	F9	FF
					!	^	?	"	ｾ	ﾌ	ﾙ	ﾟ				

（凡例）

(SP)：スペース文字です。

(4) JIS コードから EBCDIK コードへの変換

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	10	40	F0	7C	D7	79	76	20	30	57	58	91	A5	B2	DC
			(SP)	0	@	P	`	p				ｰ	ﾀ	ﾐ		
1	01	11	4F	F1	C1	D8	59	77	21	31	41	81	92	A6	B3	DD
			!	1	A	Q	a	q			｡	ｱ	ﾁ	ﾑ		
2	02	12	7F	F2	C2	D9	62	78	22	1A	42	82	93	A7	B4	DE
			"	2	B	R	b	r			｢	ｲ	ﾂ	ﾒ		
3	03	13	7B	F3	C3	E2	63	80	23	33	43	83	94	A8	B5	DF
			#	3	C	S	c	s			｣	ｳ	ﾃ	ﾓ		
4	37	3C	E0	F4	C4	E3	64	8B	24	34	44	84	95	A9	B6	EA
			$	4	D	T	d	t			､	ｴ	ﾄ	ﾔ		
5	2D	3D	6C	F5	C5	E4	65	9B	25	35	45	85	96	AA	B7	EB
			%	5	E	U	e	u			･	ｵ	ﾅ	ﾕ		
6	2E	32	50	F6	C6	E5	66	9C	06	36	46	86	97	AC	B8	EC
			&	6	F	V	f	v			ｦ	ｶ	ﾆ	ﾖ		
7	2F	26	7D	F7	C7	E6	67	A0	17	08	47	87	98	AD	B9	ED
			'	7	G	W	g	w			ｧ	ｷ	ﾇ	ﾗ		
8	16	18	4D	F8	C8	E7	68	AB	28	38	48	88	99	AE	CA	EE
			(	8	H	X	h	x			ｨ	ｸ	ﾈ	ﾘ		
9	05	19	5D	F9	C9	E8	69	B0	29	39	49	89	9A	AF	CB	EF
			)	9	I	Y	i	y			ｩ	ｹ	ﾉ	ﾙ		
A	15	3F	5C	7A	D1	E9	70	B1	2A	3A	51	8A	9D	BA	CC	FA
			*	:	J	Z	j	z			ｪ	ｺ	ﾊ	ﾚ		
B	0B	27	4E	5E	D2	4A	71	C0	2B	3B	52	8C	9E	BB	CD	FB
			+	;	K	[	k	{			ｫ	ｻ	ﾋ	ﾛ		
C	0C	1C	6B	4C	D3	5B	72	6A	2C	04	53	8D	9F	BC	CE	FC
			,	<	L	¥	l	|			ｬ	ｼ	ﾌ	ﾜ		
D	0D	1D	60	7E	D4	5A	73	D0	09	14	54	8E	A2	BD	CF	FD
			-	=	M	]	m	}			ｭ	ｽ	ﾍ	ﾝ		
E	0E	1E	4B	6E	D5	5F	74	A1	0A	3E	55	8F	A3	BE	DA	FE
			.	>	N	^	n	~			ｮ	ｾ	ﾎ	ﾞ		
F	0F	1F	61	6F	D6	6D	75	07	1B	E1	56	90	A4	BF	DB	FF
			/	?	O	_	o				ｯ	ｿ	ﾏ	ﾟ		

（凡例）

(SP)：スペース文字です。

(5) IBM_EBCDIC コードから JIS コードへの変換

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	10			20	26	2D						7B	7D	5C	30
					(SP)	&	−						{	}	¥	0
1	01	11					2F		61	6A	7E		41	4A		31
							/		a	j	~		A	J		1
2	02	12		16					62	6B	73		42	4B	53	32
									b	k	s		B	K	S	2
3	03	13							63	6C	74		43	4C	54	33
									c	l	t		C	L	T	3
4									64	6D	75		44	4D	55	34
									d	m	u		D	M	U	4
5	09		0A						65	6E	76		45	4E	56	35
									e	n	v		E	N	V	5
6		08	17						66	6F	77		46	4F	57	36
									f	o	w		F	O	W	6
7	7F		1B	04					67	70	78		47	50	58	37
									g	p	x		G	P	X	7
8		18							68	71	79		48	51	59	38
									h	q	y		H	Q	Y	8
9		19						60	69	72	7A		49	52	5A	39
								`	i	r	z		I	R	Z	9
A					5B	5D	7C	3A								
					[	]	|	:								
B	0B				2E	24	2C	23								
					.	$	,	#								
C	0C	1C		14	3C	2A	25	40								
					<	*	%	@								
D	0D	1D	05	15	28	29	5F	27								
					(	)	_	'								
E		1E	06		2B	3B	3E	3D								
					+	;	>	=								
F		1F	07	1A	21	5E	3F	22								
					!	^	?	"								

（凡例）

(SP)：スペース文字です。

（6） JIS コードから IBM_EBCDIC コードへの変換

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	10	40	F0	7C	D7	79	97								
			(SP)	0	@	P	`	p				ー	タ	ミ		
1	01	11	4F	F1	C1	D8	81	98								
			!	1	A	Q	a	q			。	ア	チ	ム		
2	02	12	7F	F2	C2	D9	82	99								
			"	2	B	R	b	r			「	イ	ツ	メ		
3	03	13	7B	F3	C3	E2	83	A2								
			#	3	C	S	c	s			」	ウ	テ	モ		
4	37	3C	5B	F4	C4	E3	84	A3								
			$	4	D	T	d	t			、	エ	ト	ヤ		
5	2D	3D	6C	F5	C5	E4	85	A4								
			%	5	E	U	e	u			・	オ	ナ	ユ		
6	2E	32	50	F6	C6	E5	86	A5								
			&	6	F	V	f	v			ヲ	カ	ニ	ヨ		
7	2F	26	7D	F7	C7	E6	87	A6								
			'	7	G	W	g	w			ァ	キ	ヌ	ラ		
8	16	18	4D	F8	C8	E7	88	A7								
			(	8	H	X	h	x			ィ	ク	ネ	リ		
9	05	19	5D	F9	C9	E8	89	A8								
			)	9	I	Y	i	y			ゥ	ケ	ノ	ル		
A	25	3F	5C	7A	D1	E9	91	A9								
			*	:	J	Z	j	z			ェ	コ	ハ	レ		
B	0B	27	4E	5E	D2	4A	92	C0								
			+	;	K	[	k	{			ォ	サ	ヒ	ロ		
C	0C	1C	6B	4C	D3	E0	93	6A								
			,	<	L	¥	l	|			ャ	シ	フ	ワ		
D	0D	1D	60	7E	D4	5A	94	D0								
			-	=	M	]	m	}			ュ	ス	ヘ	ン		
E		1E	4B	6E	D5	5F	95	A1								
			.	>	N	^	n	~			ョ	セ	ホ	゛		
F		1F	61	6F	D6	6D	96	07								
			/	?	O	_	o				ッ	ソ	マ	゜		

（凡例）

(SP)：スペース文字です。

（7） IBM_EBCDIK コードから JIS コードへの変換

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	10			20	26	2D			BF					24	30
					(SP)	&	−			ｿ					$	0
1	01	11			A1	AA	2F		B1	C0	7E		41	4A		31
					｡	ｪ	/		ｱ	ﾀ	~		A	J		1
2	02	12		16	A2	AB			B2	C1	CD		42	4B	53	32
					「	ｫ			ｲ	ﾁ	ﾍ		B	K	S	2
3	03	13			A3	AC			B3	C2	CE		43	4C	54	33
					」	ｬ			ｳ	ﾂ	ﾎ		C	L	T	3
4					A4	AD			B4	C3	CF		44	4D	55	34
					、	ｭ			ｴ	ﾃ	ﾏ		D	M	U	4
5	09		0A		A5	AE			B5	C4	D0		45	4E	56	35
					・	ｮ			ｵ	ﾄ	ﾐ		E	N	V	5
6		08	17		A6	AF			B6	C5	D1		46	4F	57	36
					ヲ	ｯ			ｶ	ﾅ	ﾑ		F	O	W	6
7	7F		1B	04	A7				B7	C6	D2		47	50	58	37
					ｧ				ｷ	ﾆ	ﾒ		G	P	X	7
8		18			A8	B0			B8	C7	D3		48	51	59	38
					ｨ	ｰ			ｸ	ﾇ	ﾓ		H	Q	Y	8
9		19			A9				B9	C8	D4		49	52	5A	39
					ｩ				ｹ	ﾈ	ﾔ		I	R	Z	9
A					5B	21		3A	BA	C9	D5	DA				
					[	!		:	ｺ	ﾉ	ｭ	ﾚ				
B	0B				2E	5C	2C	23				DB				
					.	¥	,	#				ﾛ				
C	0C	1C		14	3C	2A	25	40	BB		D6	DC				
					<	*	%	@	ｻ		ﾖ	ﾜ				
D	0D	1D	05	15	28	29	5F	27	BC	CA	D7	DD				
					(	)	_	'	ｼ	ﾊ	ﾗ	ﾝ				
E	0E	1E	06		2B	3B	3E	3D	BD	CB	D8	DE				
					+	;	>	=	ｽ	ﾋ	ﾘ	ﾞ				
F	0F	1F	07	1A	7C	5E	3F	22	BE	CC	D9	DF				
					|	^	?	"	ｾ	ﾌ	ﾙ	ﾟ			

（凡例）

(SP)：スペース文字です。

（8） JIS コードから IBM_EBCDIK コードへの変換

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	10	40	F0	7C	D7						58	91	A5		
			(SP)	0	@	P	`	p				ｰ	ﾀ	ﾐ		
1	01	11	5A	F1	C1	D8					41	81	92	A6		
			!	1	A	Q	a	q			｡	ｱ	ﾁ	ﾑ		
2	02	12	7F	F2	C2	D9					42	82	93	A7		
			"	2	B	R	b	r			｢	ｲ	ﾂ	ﾒ		
3	03	13	7B	F3	C3	E2					43	83	94	A8		
			#	3	C	S	c	s			｣	ｳ	ﾃ	ﾓ		
4	37	3C	E0	F4	C4	E3					44	84	95	A9		
			$	4	D	T	d	t			､	ｴ	ﾄ	ﾔ		
5	2D	3D	6C	F5	C5	E4					45	85	96	AA		
			%	5	E	U	e	u			･	ｵ	ﾅ	ﾕ		
6	2E	32	50	F6	C6	E5					46	86	97	AC		
			&	6	F	V	f	v			ｦ	ｶ	ﾆ	ﾖ		
7	2F	26	7D	F7	C7	E6					47	87	98	AD		
			'	7	G	W	g	w			ｧ	ｷ	ﾇ	ﾗ		
8	16	18	4D	F8	C8	E7					48	88	99	AE		
			(	8	H	X	h	x			ｨ	ｸ	ﾈ	ﾘ		
9	05	19	5D	F9	C9	E8					49	89	9A	AF		
			)	9	I	Y	i	y			ｩ	ｹ	ﾉ	ﾙ		
A	25	3F	5C	7A	D1	E9					51	8A	9D	BA		
			*	:	J	Z	j	z			ｪ	ｺ	ﾊ	ﾚ		
B	0B	27	4E	5E	D2	4A					52	8C	9E	BB		
			+	;	K	[	k	{			ｫ	ｻ	ﾋ	ﾛ		
C	0C	1C	6B	4C	D3	5B		4F			53	8D	9F	BC		
			,	<	L	¥	l	|			ｬ	ｼ	ﾌ	ﾜ		
D	0D	1D	60	7E	D4						54	8E	A2	BD		
			-	=	M	]	m	}			ｭ	ｽ	ﾍ	ﾝ		
E	0E	1E	4B	6E	D5	5F		A1			55	8F	A3	BE		
			.	>	N	^	n	~			ｮ	ｾ	ﾎ	ﾞ		
F	0F	1F	61	6F	D6	6D		07			56	90	A4	BF		
			/	?	O	_	o				ｯ	ｿ	ﾏ	ﾟ		

（凡例）

(SP)：スペース文字です。

付録 L　用語解説

Interschema で使用する用語について説明します。

(英字)

ANSI X12

ANSI（American National Standards Institute：アメリカ規格協会）が定めたアメリカ国内標準規格です。

CII シンタックスルール

CII（Center for the Informatization of Industry：産業情報化推進センター）が定めたシンタックスルールです。EIAJ 標準を基に作成され，基本的に EIAJ シンタックスルールを内包する内容です。CII シンタックスルールと各業界が開発した標準メッセージとを合わせて CII 標準と呼びます。

DTD

Document Type Definition の略です。XML 文書で使用するタグの属性や階層構造や出現順序などを定義したものです。XML 文書はこの DTD で定義した文法に従って記述します。

EDI

EDI（Electronic Data Interchange：電子データ交換）とは，「異なる企業間で，商取引のためのデータを，通信回線を介して標準的な規約（できる限り広く合意された各種規約）を用いて，コンピュータ（端末も含む）間で交換すること」（EDI 推進協議会（JEDIC：Japan Electronic Data Interchange Council））を意味します。

EDIFACT

EDIFACT（Electronic Data Interchange for Administration, Commerce and Transport：行政，商業，運輸のための電子データ交換に関する規格）は，ISO に登録された EDI シンタックスルールです。

EIAJ 標準

EIAJ（Electronic Industries Association of Japan：日本電子機械工業会）が定めた同業界の EDI 標準規格です。日本で初めて可変長フォーマットを採用したシンタックスルールです。

EJB（Enterprise Java Beans）

Java でサーバサイドのプログラムを作成するときに使用するアプリケーションモデルです。

EMPTY 要素

要素に内容がない場合を示します。

FDL ファイル

FDL エディタで定義したファイルです。ファイルの拡張子は，「.fdl」です。

Java VM（Virtual Machine）

Java コンパイラを使って生成した中間コードによるプログラムを解釈・実行する環境です。

JEDICOS

JEDICOS（Japan EDI for Commerce Systems）は，財団法人 流通システム開発センターによって定められた国内流通業界での EDI の標準規格です。EDI シンタックスルールは EDIFACT に従います。

JNI（Java Native Interface）

Java からプラットフォーム固有の機能を呼び出す際に利用する機能です。

MDL ファイル

変換する情報を定義したファイルです。ファイルの拡張子は，「.mdl」です。

Sea-NACCS

Sea-NACCS（Sea-Nippon Automated Cargo Clearance System：海上貨物通関情報処理システム）は，厚生労働省や財務省などの各省庁と輸出入関連の民間業界をネットワークで接続し，通関に関する一連の手続きを処理するシステムです。このシステムで扱う電文には，Sea-NACCS 形式の EDI 電文（Sea-NACCS EDI 電文）と EDIFACT 形式の電文（Sea-NACCS EDIFACT 電文）があります。

このシステムの稼働開始及び更改の時期は次のとおりです。

- 1999 年 10 月
  システムの稼働を開始
- 2008 年 10 月
  サービスレベルの維持と向上，国際物流の情報化推進などを目的として，システムを更改

Servlet

Java で Web のコンポーネントを構築するためのインタフェースです。

TFD

TFD（Transfer Form Data）は CII 標準又は EIAJ 標準で可変長データエレメントを扱う形式で，データタグ（データエレメントの番号を表す），レングスタグ（データエレメントの長さを表す），データエレメント（データ本体）から構成されます。

UN/EDIFACT

UN/EDIFACT（United Nations Rules for Electronic Data Interchange for Administration, Commerce and Transport：行政，商業，運輸のための電子データ交換に関する国連規格）は，国際連合によって定められた国際的な EDI 標準規格です。

white space

XML データのタブ，改行，復帰，スペースの各文字の任意の組み合わせのことです。

XML

Extensible Markup Language の略です。W3C が制定したマークアップ言語で，インターネット上でデータのやり取りを実現します。このマニュアルでは，XML 形式で記述された文書を XML 文書と呼びます。

XML 文書

XML 形式で記述された文書です。

（ア行）

アイテム

画面上（FDM エディタ及び MDL エディタ）のツリー（木構造形式の GUI 部品）やリスト（表形式の GUI 部品）内に表示されるオブジェクトです。型，セパレータ，変数，構造，XML 属性などを表します。

（カ行）

外部サブセット

XML の文書型宣言の外部サブセットは，文書型宣言の外部パラメタ実体で宣言される，名前の付けられない外部実体に格納される DTD 宣言の集合を指します。外部実体とは，その宣言が格納されている場所とは，別の外部に内容が格納されている実体のことです。

型

データ中の項目となる要素の属性，サイズや要素の受け取る値などを定義したものです。

空要素

内容がない要素のことです。XML では，開始タグの直後に終了タグを置くか，空要素タグを使って表します。空要素タグは，開始タグの末尾にある「＞」の直前に「/」を挿入したものです。

グローバル名

MDL ファイル内で使用するコンポーネント（構造又は要素）の名前です。フォーマット名から目的とするコンポーネント名までを「@」でつないだ文字列です。

構造

変換するデータの構成を表す概念で，子コンポーネントを持ちます。C 言語の構造体に相当します。

固定長方式

データ要素が決められた長さを持つ方式です。

コメント

XML のコメントとは，プログラミング言語の場合と同じように，そのソースを読む人に向けられたメッセージのことです。

コンポーネント

エディタで定義された構造の構成メンバです。コンポーネントは構造又は型定義された要素です。

（サ行）

セパレータ

データを区切る記号です。CSV フォーマットのコンマや改行などが該当します。

（タ行）

デリミタ (セパレータ) 方式

連続するデータ要素の区切りにデリミタ（セパレータ）を使用して可変長データを扱う方式です。UN/EDIFACT，ANSI X12 などが該当します。

（ナ行）

内容モデル

XML の子要素の型及び子要素の出現順序を制御する簡単な文法のことです。要素型宣言の一部として，ある型の要素としてありえる内容を記述します。

（ハ行）

フォーマット

変換するデータの構造を定義したものです。FDL エディタで定義します。

（マ行）

メッセージ

EDI で交換される情報の単位です。通常，1 件の取引が一つのメッセージになります。

（ヤ行）

要素

FDL ファイル／ MDL ファイルでは変換するデータの末端の構成要素で，構造の子供に相当する概念です。変換するデータ中の 1 項目に相当します。

XML の場合は，文書の型ごとに定義される論理構造の 1 単位に相当します。要素は，開始タグと終了タグによって表現されます。

（ラ行）

レングスタグ方式

データ要素に対して，その長さ又は個数を表す数値情報を付けることで可変長データを扱う方式です。CII 標準，EIAJ 標準などが該当します。

ローカル名

FDL ファイル内で使用するコンポーネント（構造又は要素）の名前です。

索引

記号

数字

A

C

D

E

F

S

T

U

V

W

X

あ

い

う

え

か

き

く

け

こ

さ

し

す

せ

そ

た

ち

つ

て

と

な

に

は

ひ

ふ

へ

ま

み

む

め

も

ゆ

よ

ら

り

る

れ

ろ

ソフトウェアマニュアルのサービス ご案内

１．マニュアル情報ホームページ

ソフトウェアマニュアルの情報をインターネットで公開しています。

URL　http://www.hitachi.co.jp/soft/manual/

ホームページのメニューは次のとおりです。

■マニュアル一覧	日立コンピュータ製品マニュアルを製品カテゴリ，マニュアル名称，資料番号のいずれかから検索できます。
■CD-ROMマニュアル	日立ソフトウェアマニュアルと製品群別CD-ROMマニュアルの仕様について記載しています。
■マニュアルのご購入	マニュアルご購入時のお申し込み方法を記載しています。
■オンラインマニュアル	一部製品のマニュアルをインターネットで公開しています。
■サポートサービス	ソフトウェアサポートサービスお客様向けページでのマニュアル公開サービスを記載しています。
■ご意見・お問い合わせ	マニュアルに関するご意見，ご要望をお寄せください。

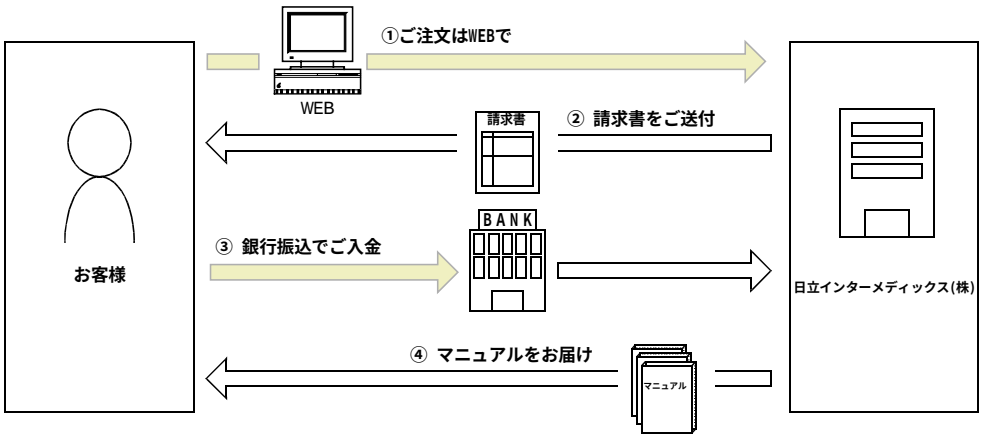
２．インターネットでのマニュアル公開

2 種類のマニュアル公開サービスを実施しています。

(1) マニュアル情報ホームページ「オンラインマニュアル」での公開
　製品をよりご理解いただくためのご参考として，一部製品のマニュアルを公開しています。

(2) ソフトウェアサポートサービスお客様向けページでのマニュアル公開
　ソフトウェアサポートサービスご契約のお客様向けにマニュアルを公開しています。公開しているマニュアルの一覧，本サービスの対象となる契約の種別などはマニュアル情報ホームページの「サポートサービス」をご参照ください。

３．マニュアルのご注文



① マニュアル情報ホームページの「マニュアルのご購入」にアクセスし，お申し込み方法をご確認のうえ WEB からご注文ください。ご注文先は日立インターメディックス(株)となります。

② ご注文いただいたマニュアルについて請求書をお送りします。

③ 請求書の金額を指定銀行へ振り込んでください。

④ 入金確認後 7 日以内にお届けします。在庫切れの場合は，納期を別途ご案内いたします。