

MISP 基本機能アクセスツール

2007 年 1 月 22 日

目次

1	コマンドの使用方法	1
1.1	コマンド種類と用途	1
1.2	DarumaClient コマンド	2
1.3	ShowFeatureTypes コマンド	2
1.4	RegisterFeatureType コマンド	3
1.5	DescribeFeatureType コマンド	3
1.6	CountFeatureType コマンド	4
1.7	XmlGetFeature コマンド	5
1.8	XmlInsert コマンド	6
1.9	CsvGetFeature コマンド	8
1.10	CsvInsert コマンド	9
1.11	-debug オプションの使用について	13

基本機能アクセスツールの概要

基本機能アクセスツールは、DaRuMa が使用している減災情報共有プロトコルである MISP を使用したクライアントツールである。本ツールで提供するコマンド群は、GetFeature、Insert、Update、Delete 等の基本リクエストに加え、登録されている FeatureType のリストや指定した FeatureType の登録データ数を取得するリクエストを処理することができる。

1 コマンドの使用方法

1.1 コマンド種類と用途

基本機能アクセスツールが提供するコマンドは、DarumaClient、RegisterFeatureType、DescribeFeatureType、CountFeatureType、ShowFeatureTypes、XmlGetFeature、XmlInsert、CsvGetFeature、CsvInsert である。

1.2 DarumaClient コマンド

DarumaClient コマンドは、MISP に準拠した XML リクエストを送信し、レスポンスを受信するクライアントとして動作する。XML リクエストは、ファイルまたは標準入力を使用して指定できる。DarumaClient コマンドは、汎用コマンドであるため、GetFeature や Insert、Update、Delete 等の MISP のリクエストファイルを生成することによって、様々な用途に使用できる。

```
usage: DarumaClient [options] -host HOSTNAME -in INPUT_FILE
  -debug                中間ファイルを削除しません。stacktrace を出力します。
  -host <HOSTNAME>      サーバの IP アドレスまたはホスト名
  -in <INPUT_FILE>      入力ファイル名 (XML ファイル)
  -notaddheader          リクエストは SOAP の HEADER 部分を追加しません。
  -onlybody              レスポンスは SOAP の BODY 部分のみを出力します。
  -out <OUTPUT_FILE>    出力ファイル名 (XML ファイル)
  -xpath                XPath による出力モード
```

request.xml を送信して結果をコンソール出力する場合の例

```
DarumaClient -host 127.0.0.1 -in request.xml
DarumaClient -host 127.0.0.1 < request.xml
```

request.xml を送信して結果 I を response.xml に出力する場合の例

```
DarumaClient -host 127.0.0.1 -in request.xml -out response.xml
DarumaClient -host 127.0.0.1 -in request.xml > response.xml
```

SOAP 本体部分のみを出力結果とする場合の例

```
DarumaClient -host 127.0.0.1 -onlybody -in request.xml -out response.xml
```

XPATH 形式で結果を出力する場合の例

```
DarumaClient -host 127.0.0.1 -onlybody -xpath -in request.xml
```

DarumaClient コマンドのパラメータ

```
darumaclient.retrieve.soap.body.xslt.template SOAP ボディ部分を抽出する XSLT ファイル
```

1.3 ShowFeatureTypes コマンド

ShowFeatureType コマンドは、現在、サーバに登録されている全 FeatureType を取得するクライアントとして動作する。ShowFeatureTypes は、MISP の GetCapabilities プロトコルを使用しており、このリクエストのレスポンスから FeatureType の名前を取得している。

```
usage: ShowFeatureTypes [options] -host HOSTNAME
-debug                中間ファイルを削除しません。stacktrace を出力します。
-host <HOSTNAME>      サーバのホスト名又は IP アドレス
-out <OUTPUT_FILE>    出力ファイル名 (XML ファイル)
```

FeatureType のリストをコンソール出力する場合の例

```
ShowFeatureType -host 127.0.0.1
```

FeatureType のリストを response.txt に出力する場合の例

```
ShowFeatureType -host 127.0.0.1 > response.txt
```

ShowFeatureTypes コマンドのパラメータ

```
showfeaturetypes.wfs2text.xslt.template FeatureType のリストを抽出する XSLT ファイル
```

1.4 RegisterFeatureType コマンド

RegisterFeatureType コマンドは、指定した FeatureType のスキーマを登録するクライアントとして動作する。RegisterFeatureType は、MISP の RegisterFeatureType プロトコルを使用しており、MISP の RegisterFeatureType と同様の結果となる。

```
usage: RegisterFeatureType [options] -host HOSTNAME -in INPUT_FILE
-debug                中間ファイルを削除しません。stacktrace を出力します。
-host <HOSTNAME>      サーバの IP アドレスまたはホスト名
-in <INPUT_FILE>      入力ファイル名 (XSD ファイル)
-out <OUTPUT_FILE>    出力ファイル名 (XML ファイル)
```

FeatureType を登録して結果をコンソール出力する場合の例

```
RegisterFeatureType -host 127.0.0.1 -in Building.xsd
```

FeatureType を登録して結果を response.xml する場合の例

```
RegisterFeatureType -host 127.0.0.1 -in Building.xsd -out response.xml
RegisterFeatureType -host 127.0.0.1 -in Building.xsd > response.xml
```

1.5 DescribeFeatureType コマンド

DescribeFeatureType コマンドは、指定した FeatureType のスキーマを出力するクライアントとして動作する。DescribeFeatureType は、MISP の DescribeFeatureType プロトコルを使用しており、-xpath オプションをしていない場合は、MISP の DescribeFeatureType と同様の結果となる。

```
usage: DescribeFeatureType [options] -host HOSTNAME {名前空間}Feature 名
-debug                        中間ファイルを削除しません。stacktrace を出力します。
-host <HOSTNAME>            サーバの IP アドレスまたはホスト名
-out <OUTPUT_FILE>          出力ファイル名 (XML ファイル)
-xpath                       XPath による出力モード
```

FeatureType を指定して結果をコンソール出力する場合の例

```
DescribeFeatureType -host 127.0.0.1 {http://...}Building
```

FeatureType を指定して結果を response.xml する場合の例

```
DescribeFeatureType -host 127.0.0.1 -out response.xml {http://...}Building
DescribeFeatureType -host 127.0.0.1 {http://...}Building > response.xml
```

XPath 形式で結果を表示する場合の例

```
DescribeFeatureType -xpath -host 127.0.0.1 {http://...}Building
```

DescribeFeatureType コマンドのパラメータ

```
describefeaturetype.retrieve.schema.xslt.template  Schema 定義を抽出する XSLT ファイル
```

1.6 CountFeatureType コマンド

CountFeatureType は、指定した FeatureType の登録件数を返すクライアントとして動作する。出力結果として、登録件数の数字のみを表示する。カウントされる FeatureType は、サーバに登録されている同一の FeatureType の全データである。^{*1}

```
usage: CountFeatureType [options] -host HOSTNAME {名前空間}Feature 名
-debug                        中間ファイルを削除しません。stacktrace を出力します。
-host <HOSTNAME>            サーバの IP アドレスまたはホスト名
-out <OUTPUT_FILE>          出力ファイル名 (XML ファイル)
```

FeatureType を指定して登録件数をコンソール出力する場合の例

```
CountFeatureType -host 127.0.0.1 {http://...}Building
```

CountFeatureType コマンドのパラメータ

```
countfeaturetype.create.getfeature.count.xslt.template リクエスト生成用 XSLT ファイル
countfeaturetype.retrieve.count.xslt.template 件数を抽出する XSLT ファイル
```

^{*1} 現在、サーバ側が未実装のため動作テストを行っていない。

1.7 XmlGetFeature コマンド

XmlGetFeature は、MISP の GetFeature プロトコルを用いて、データを取得するクライアントとして動作する。

```
usage: XmlGetFeature [options] -host HOSTNAME
  -checkFeature <FEATURE>   Feature を「{名前空間}Feature 名」形式で指定する。
  -debug                     中間ファイルを削除しません。stacktrace を出力します。
  -diff                      差分データを取得
  -host <HOSTNAME>          サーバの IP アドレスまたはホスト名
  -in <INPUT_FILE>          入力ファイル名 (MISP クエリファイル)
  -interval <INTERVAL>      ループ時間間隔。
  -loop                      ループで動作を繰り返す。
  -out <OUTPUT_FILE>        出力ファイル名 (XML ファイル)
```

getfeature.xml を送信して結果をコンソール出力する場合の例

```
XmlGetFeature -host 127.0.0.1 -in getfeature.xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<misp:GetFeature xmlns="http://.../Rescue/RandomCity/2.0"
  xmlns:rcbase="http://staff.aist.go.jp/i.noda/Rescue/RandomCity/base"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:gml="http://www.opengis.net/gml"
  xmlns:misp="http://www.infosharp.org/misp">
  <misp:Query typeName="Building">
    <misp:Filter>
      <misp:True/>
    </misp:Filter>
  </misp:Query>
</misp:GetFeature>
```

FeatureType を指定して結果をコンソール出力する場合の例

```
XmlGetFeature -host 127.0.0.1 -checkFeature {http://...}Building
```

FeatureType を指定して結果を data.xml ファイルに出力する場合の例

```
XmlGetFeature -host 127.0.0.1 -checkFeature {http://...}Building -out data.xml
XmlGetFeature -host 127.0.0.1 -checkFeature {http://...}Building > data.xml
```

定期的に (5 秒)FeatureType を指定して結果をコンソールに出力する場合の例

```
XmlGetFeature -host 127.0.0.1 -loop -interval 5 -checkFeature {http://...}Building
```

定期的に (10 秒)FeatureType を指定して結果を data.xml ファイルに出力する場合の例

```
XmlGetFeature -host 127.0.0.1 -loop -checkFeature {http://...}Building -out data.xml
```

定期的に (10 秒)FeatureType を指定して 差分データを data.xml ファイルに出力する場合の例

```
XmlGetFeature -host 127.0.0.1 -loop -diff -checkFeature {http://...}Building -out data.xml
```

出力されるファイル名は data.xml.ZZZZZZZ(00000001 ~ 99999999) になります。

1.8 XmlInsert コマンド

XmlInsert は、MISP の Insert プロトコルを用いて、XML ファイルのデータを挿入するクライアントとして動作する。

```
usage: XmlInsert [options] -host HOSTNAME -in INPUT_FILE
  -debug                中間ファイルを削除しません。stacktrace を出力します。
  -host <HOSTNAME>      サーバの IP アドレスまたはホスト名
  -in <INPUT_FILE>      入力ファイル名 (XML ファイル)
  -inDir <IN_DIR>       監視するフォルダ。
  -inPattern <IN_PATTERN> 入力ファイルの正規表現
  -interval <INTERVAL>   ループ時間間隔。
  -loop                 ループで動作を繰り返す。
  -out <OUTPUT_FILE>     出力ファイル名 (XML ファイル)
  -trash <TRASH>        処理の終わった入力ファイルの格納先のフォルダ
  -xpath                XPath による出力モード
```

Data.xml を送信して結果をコンソール出力する場合の例

```
XmlInsert -host 127.0.0.1 -in Data.xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<misp:FeatureCollection xmlns:gml="http://www.opengis.net/gml"
xmlns:misp="http://www.infosharp.org/misp"
xmlns:rcbase="http://.../Rescue/RandomCity/base">
  <gml:featureMember>
    <Building xmlns="http://staff.aist.go.jp/i.noda/Rescue/RandomCity/2.0">
      .....
    </Building>
  </gml:featureMember>
  <Building xmlns="http://staff.aist.go.jp/i.noda/Rescue/RandomCity/2.0">
    .....
  </Building>
<gml:featureMember>
</gml:featureMember>
.....
</misp:FeatureCollection>
```

Data.xml を送信して結果を response.xml ファイルに出力する場合の例

```
XmlInsert -host 127.0.0.1 -in Data.xml -out response.xml
XmlInsert -host 127.0.0.1 -in Data.xml > response.xml
```

Data.xml を送信して SOAP のボディのみを response.xml ファイルに出力する場合の例

```
XmlInsert -host 127.0.0.1 -onlybody -in Data.xml -out response.xml
XmlInsert -host 127.0.0.1 -onlybody -in Data.xml > response.xml
```

Data.xml を送信して SOAP のボディのみ XPath 形式で response.xml ファイルに出力する場合の例

```
XmlInsert -host 127.0.0.1 -onlybody -xpath -in Data.xml -out response.xml
XmlInsert -host 127.0.0.1 -onlybody -xpath -in Data.xml > response.xml
```

定期的に (5 秒) data フォルダを監視し、.xml ファイルを送信し、結果を trash フォルダに保存する例

```
XmlInsert -host 127.0.0.1 -loop -interval 5 -inDir .\data -inPattern *.xml -trash .\trash
```

-inPattern のパラメータは正規表現で指定します。

正常処理されたファイルは trash フォルダに移動されます。

結果は trash\result フォルダに保存されます。

処理不正終了のファイルは trash\error フォルダに移動されます

1.9 CsvGetFeature コマンド

CsvGetFeature は、MISP の GetFeature プロトコルに従って取得したデータを、CSV に変換するクライアントとして動作する。CSV への変換ルールは-csvcfg オプションで指定されたファイルに従って変換を行う。

```
usage: CsvGetFeature [options] -host HOSTNAME -csvcfg CONFIG_FILE
  -encode <ENCODE>           出力 CSV エンコード (UTF-8/Shift_JIS/EUC-JP)
  -checkFeature <FEATURE>    Feature を「{名前空間}Feature 名」形式で指定する。
  -csvcfg <CONFIG_FILE>      スキーマ CSV マッピング定義ファイル
  -debug                      中間ファイルを削除しません。stacktrace を出力します。
  -diff                      差分データを取得
  -host <HOSTNAME>           サーバの IP アドレスまたはホスト名
  -in <INPUT_FILE>           入力ファイル名 (MISP クエリファイル)
  -interval <INTERVAL>      ループ時間間隔。
  -loop                      ループで動作を繰り返す。
  -out <OUTPUT_FILE>         出力ファイル名 (CSV ファイル)
```

FeatureType を指定して 結果をコンソール出力する場合の例

CSV の変換ルール定義ファイルが map.xml の場合

```
CsvGetFeature -host 127.0.0.1 -csvcfg map.xml -checkFeature {名前空間}Feature 名
```

結果を Shift_JIS で出力する場合

```
CsvGetFeature -host 127.0.0.1 -encode Shift_JIS -csvcfg map.xml -checkFeature {名前空間}Feature 名
```

結果を EUC-JP で出力する場合

```
CsvGetFeature -host 127.0.0.1 -encode EUC-JP -csvcfg map.xml -checkFeature {名前空間}Feature 名
```

getfeature.xml を送信して結果を response.csv ファイルに出力する場合の例

CSV の変換ルール定義ファイルが map.xml の場合

```
CsvGetFeature -host 127.0.0.1 -csvcfg map.xml -in getfeature.xml -out response.csv
```

```
CsvGetFeature -host 127.0.0.1 -csvcfg map.xml -in getfeature.xml > response.csv
```

定期的に (10 秒)FeatureType を指定して結果をコンソールに出力する場合の例

CSV の変換ルール定義ファイルが map.xml の場合

```
CsvGetFeature -host 127.0.0.1 -loop -csvcfg map.xml -checkFeature {http://...}Building
```

定期的に (10 秒)FeatureType を指定して 差分データを data.csv ファイルに出力する場合の例

```
CsvGetFeature -host 127.0.0.1 -loop -diff -csvcfg map.xml -checkFeature {http://...}Building -o
```


出力されるファイル名は data.csv.ZZZZZZZZ(00000001 ~ 99999999) になります。

1.10 CsvInsert コマンド

CsvInsert は、MISP の Insert プロトコルを用いて、CSV ファイルのデータを挿入するクライアントとして動作する。CSV の変換ルールは-csvcfg オプションで指定されたファイルに従って変換を行う。

```
usage: CsvInsert [options] -host HOSTNAME -csvcfg CONFIG_FILE
  -encode <ENCODE>           入力 CSV エンコード (UTF-8/Shift_JIS/EUC-JP)
  -csvcfg <CONFIG_FILE>      スキーマ CSV マッピング定義ファイル
  -debug                      中間ファイルを削除しません。stacktrace を出力します。
  -host <HOSTNAME>           サーバの IP アドレスまたはホスト名
  -in <INPUT_FILE>           入力ファイル名 (CSV ファイル)
  -inDir <IN_DIR>            監視するフォルダ。
  -inPattern <IN_PATTERN>    入力ファイルの正規表現
  -interval <INTERVAL>       ループ時間間隔。
  -loop                      ループで動作を繰り返す。
  -onlybody                  SOAP の BODY 部分のみを出力します。
  -out <OUTPUT_FILE>         出力ファイル名 (XML ファイル)
  -trash <TRASH>            処理の終わった入力ファイルの格納先のフォルダ
  -xpath                     XPath による出力モード
```

Data.csv を送信して結果をコンソール出力する場合の例

CSV の変換ルール定義ファイルが map.xml の場合

```
CsvInsert -host 127.0.0.1 -csvcfg map.xml -in Data.csv
```

```
CsvInsert -host 127.0.0.1 -csvcfg map.xml < Data.csv
```

入力 CSV データが Shift_JIS の場合

```
CsvInsert -host 127.0.0.1 -encode Shift_JIS -csvcfg map.xml -in Data.csv
```

入力 CSV データが EUC-JP の場合

```
CsvInsert -host 127.0.0.1 -encode EUC-JP -csvcfg map.xml -in Data.csv
```

Data.csv を送信して結果を response.xml ファイルに出力する場合の例

CSV の変換ルール定義ファイルが map.xml の場合

```
CsvInsert -host 127.0.0.1 -csvcfg map.xml -in Data.csv -out response.xml
```

```
CsvInsert -host 127.0.0.1 -csvcfg map.xml -in Data.csv > response.xml
```

Data.csv を送信して SOAP のボディのみを response.xml ファイルに出力する場合の例

CSV の変換ルール定義ファイルが map.xml の場合

```
CsvInsert -host 127.0.0.1 -onlybody -csvcfg map.xml -in Data.csv -out response.xml
```

```
CsvInsert -host 127.0.0.1 -onlybody -csvcfg map.xml -in Data.csv > response.xml
```

Data.csv を送信して SOAP のボディのみ XPath 形式で response.xml ファイルに出力する場合の例

CSV の変換ルール定義ファイルが map.xml の場合

```
CsvInsert -host 127.0.0.1 -onlybody -xpath -csvcfg map.xml -in Data.csv -out response.xml
```

```
CsvInsert -host 127.0.0.1 -onlybody -xpath -csvcfg map.xml -in Data.csv > response.xml
```

定期的に (5 秒) data フォルダを監視し、.csv ファイルを送信し、結果を trash フォルダに保存する例

```
CsvInsert -host 127.0.0.1 -loop -interval 5 -inDir .\data -inPattern *.csv -trash .\trash
```

-inPattern のパラメータは正規表現で指定します。

正常処理されたファイルは trash フォルダに移動されます。

結果は trash\result フォルダに保存されます。

処理不正終了のファイルは trash\error フォルダに移動されます

CsvInsert コマンドのパラメータ

csvinsert.retrieve.soap.body.xslt.template SOAP ボディ部分を抽出する XSLT ファイル

1.10.1 CSV 変換ルール定義

CSV 変換は、CSV 変換ルール定義ファイルを使用して変換を行う。CSV のカラムと MISP の XML の XPath 形式を一对一で記述する。以下に記述例を示す。

```
<?xml version="1.0" encoding="UTF-8"?>
<cx:Config xmlns:cx="http://staff.aist.go.jp/i.noda/Standard/2005/CsvXml">
  <cx:xml>
    <xsd:schema id="BuildingInfo.xsd">
      ... スキーマ定義 (省略)
    </xsd:schema>
  </cx:xml>
  <!-- CSV の変換ルールを定義 -->
  <cx:csv element="BuildingFireInfo" cs="," rs="\n" quoteChar='"' commentPrefix="#">
    <cx:columnList>
      <!-- XPath に MISP の構造、name に CSV の構造を定義する -->
      <cx:column xpath="id" name="id"/>
      <cx:column xpath="damage/level" name="damagelevel"/>
      <cx:column xpath="time/begin" name="begintime"/>
      <cx:column xpath="time/end" name="endtime"/>
    </cx:columnList>
  </cx:csv>
</cx:Config>
```

1.10.2 GeomteryPropertyType の表現例

CSV カラムのデータ型が GeomteryPropertyType(Point/LineString/ Polygon)の場合は、[Well-Known Text] 形式で記述します。

Point の記述フォーマットは POINT(X Y) です。

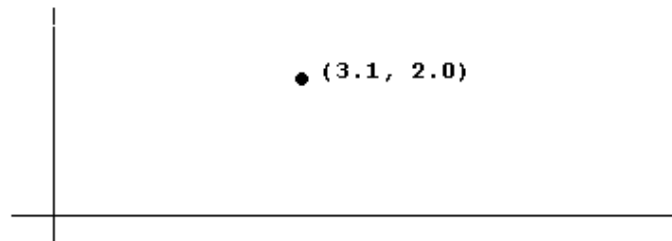
LineString の記述フォーマットは LINESTRING(X1 Y1, X2 Y2 ...) です。

Polygon の記述フォーマットは POLYGON((X1 Y1, X2 Y2 ...), (X3 Y3, X4 Y4 ...) ...) です。(X1 Y1, X2 Y2 ...) は outerBoundaryIs です。(X3 Y3, X4 Y4 ...) 及び以降の LinearRing は innerBoundaryIs になります。Polygon の outerBoundaryIs は必須項目です。innerBoundaryIs はオプション項目です。また LinearRing の起点と終点が一致する必要があります。

MultiLineString の記述フォーマットは LINESTRING(X1 Y1, X2 Y2 ...) LINESTRING(...) ... です

MultiPolygon の記述フォーマットは POLYGON((X1 Y1, X2 Y2 ...), (X3 Y3, X4 Y4 ...) ...) POLYGON(...) ... です

Point の例



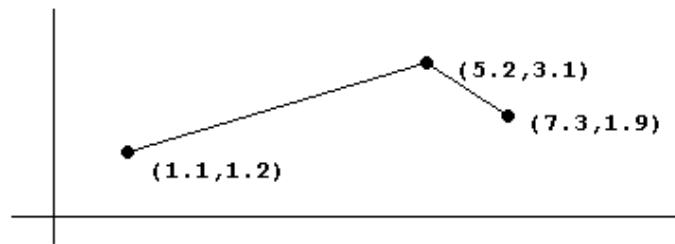
[Well-Known Text] 形式

POINT(3.1 2.0)

[XML] 形式

```
<gml:Point>
  <gml:coordinates>3.1,2.0</gml:coordinates>
</gml:Point>
```

LineString の例



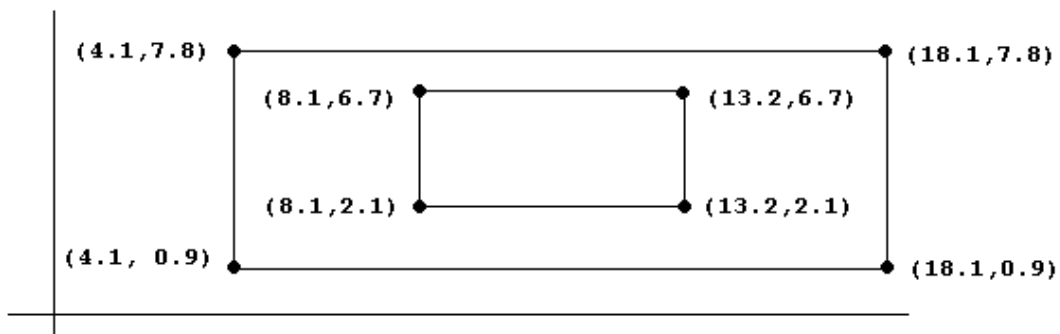
[Well-Known Text] 形式

LINESTRING(1.1 1.2,5.2 3.1,7.3 1.9)

[XML] 形式

```
<gml:LineString>
  <gml:coordinates>1.1,1.2 5.2,3.1 7.3,1.9</gml:coordinates>
</gml:LineString >
```

Polygon の例



[Well-Known Text] 形式

POLYGON((4.1 0.9,4.1 7.8,18.1 7.8,18.1 0.9,4.1 0.9),(8.1 2.1,8.1 6.7,13.2 6.7,13.2 2.1,8.1 2.1))

[XML] 形式

```
<gml:Polygon>
  <gml:outerBoundaryIs>
    <gml:LinearRing>
      <gml:coordinates>
        4.1,0.9 4.1,7.8 18.1,7.8 18.1,0.9 4.1,0.9
      </gml:coordinates>
    </gml:LinearRing>
  </gml:outerBoundaryIs>
```

```
<gml:innerBoundaryIs>
  <gml:coordinates>
    8.1,2.1 8.1,6.7 13.2,6.7 13.2,2.1 8.1,2.1
  </gml:coordinates>
</gml:innerBoundaryIs>
</gml:Polygon>
```

1.11 -debug オプションの使用について

デバックオプションを使用することにより、各コマンドツールが一時的に使用するファイルをコマンド終了後も残すことができる。一時的に使用するファイルには、DaRuMa サーバに送信する SOAP のメッセージや、レスポンスが含まれるため、障害時の問題発見に有効である。また、エラー発生箇所のスタックトレースを標準エラーに出力することができる。