

MISP 基本機能アクセスツール

NetConscious, Inc.

2006 年 9 月 21 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	1
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	4
1.8	XmlInsert コマンド	6
1.9	CsvGetFeature コマンド	7
1.10	CsvInsert コマンド	8
1.11	-debug オプションの使用について	12
2	テクニカルアーキテクチャ	13
2.1	コマンドツール群のディレクトリ構成	13
2.2	コマンド・ランタイム (アプリケーションプラットフォーム)	13
2.3	コマンド・アドオン (アドオンプログラム)	14
2.4	コマンドの実行	14
2.5	システムプロパティによるパラメータ設定	17

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。DaurmaClient 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。
-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 | 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。
  -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。
  -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。
  -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。
  -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)
```

^{*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。

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

1.7 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。
-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.ZZZZZZZZ(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。
  -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。
  -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。
  -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 GeometryPropertyType の表現例

CSV カラムのデータ型が GeometryPropertyType(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。

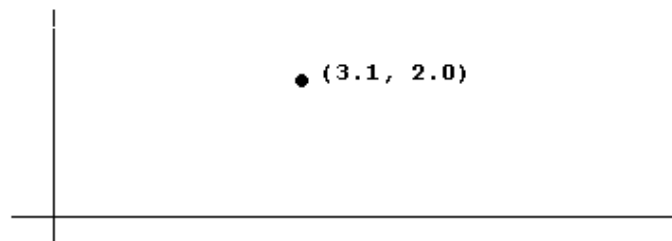
LineString の記述フォーマットは LINESTRING(X1 Y1, X2 Y2 ...) です。

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



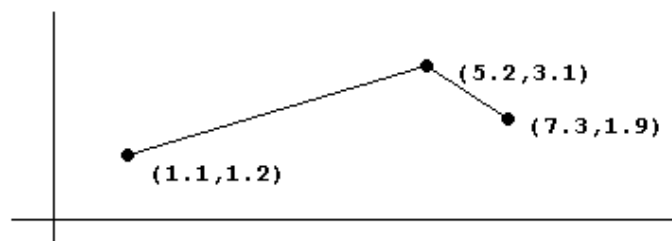
[Well-Known Text] 形式

POINT(3.1 2.0)

[XML] 形式

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

LineString の例



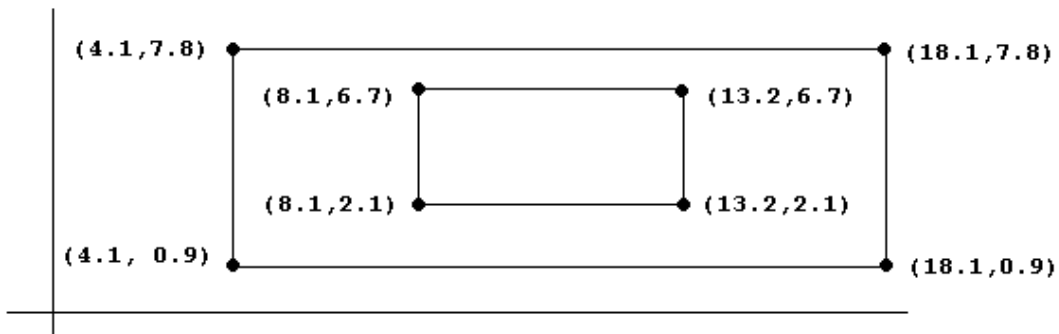
[Well-Known Text] 形式

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デバックオプションを使用することができる。

2 テクニカルアーキテクチャ

MISP 基本機能アクセスツールは、拡張性を考慮し、各コマンドに対応したコマンド・アドオンと、それを実行するコマンド・ランタイムの二つで構成されている。また、コマンド・アドオンは、Message インターフェース、Payload インターフェース、Processor インターフェースの 3 種類の Java インタフェースを基本としている。

2.1 コマンドツール群のディレクトリ構成

本ツールのディレクトリ構成は、大きく分けて、コンポーネントプラグインのための components、各種設定ファイルを格納する etc、コマンド・ランタイムのライブラリを格納する lib、コマンド・アドオン本体及びライブラリを格納する work で構成されている。以下にディレクトリ構成を示す。

<HOME>

```
components <-機能拡張を行う場合のコンポーネントを格納するディレクトリ
etc <-各コマンドの設定ファイルを格納するディレクトリ
lib <-コマンド実行環境の基本ライブラリファイルを格納するディレクトリ
work <-コマンド実行環境の作業用ディレクトリ
    beans <-コマンド実行環境をカスタマイズする場合に使用するディレクトリ
    classes <-拡張クラスを格納するためのディレクトリ
    components <-各コンポーネントが使用する作業用ディレクトリ
    lib <-コマンド実行環境の拡張ライブラリファイルを格納するディレクトリ
    log <-ログフォルダ
    samples <-サンプルデータディレクトリ
    sysprops <-各コマンドのパラメータをカスタマイズする場合に使用するディレクトリ
    xsl <-各コマンドが使用する XSLT ファイルを格納するディレクトリ
```

2.2 コマンド・ランタイム (アプリケーションプラットフォーム)

MISP 基本機能アクセスツールは、NetConscious が LGPL として開発しているアプリケーションプラットフォーム^{*2}を基に開発されている。

このアプリケーションプラットフォームは、コンポーネントプラグインアーキテクチャと DI コンテナを基本技術として実装しており、コマンドラインツールからサーバアプリケーションまでを実装することができる。今回、コンポーネントは使用していないが、ServletContainer コンポーネントを追加することにより、容易にサーバ機能を追加実装することができる。

^{*2} NetConscious BMX (BlueMeme/Basis eXpress Edition)

2.3 コマンド・アドオン (アドオンプログラム)

各コマンドは、コマンド・アドオンとして、コマンド・ランタイム上で動作するアドオンプログラムとして開発されている。共通のランタイム部分とコマンドの処理部分を分離することにより、設計の柔軟性と Java クラスの可搬性を高めている。

2.4 コマンドの実行

本ツールのコマンドは、すべてスクリプトを使用して実行することが可能であるが、Java コマンドを使用して起動することができる。各コマンドは、全てコマンド・ランタイム上で動作するコマンド・アドオンであるため、コマンド・ランタイムを起動後、コマンド・アドオンをロードして実行する必要がある。

本ツールは、起動された JavaVM のシステムプロパティから JAVA_HOME の設定値を取得するため、JAVA_HOME の設定は必須ではない。

2.4.1 コマンド・ランタイムの起動

コマンド・ランタイムを起動するには、ランタイムのホームディレクトリをシステムプロパティで指定し、startup.jar を -jar オプションを指定して Java コマンドを実行する必要がある。ランタイムのホームディレクトリを省略した場合は、Java VM を起動したカレントディレクトリをホームとして設定する。以下に、コマンド・ランタイムの起動方法と示す。

```
set DARUMA_HOME=c:\darumatool
java -Dapplication.home="%DARUMA_HOME:\=/" -jar startup.jar
```

コマンドを実行するカレントディレクトリが本ツールのホームである場合は、-Dapplication.home は省略可能である。

2.4.2 コマンド・アドオンのロード

コマンド・ランタイム単体を起動した場合、etc ディレクトリ以下の default.system.properties.xml を読み込み標準の StandardApplication クラス (標準のコマンド・アドオン) が起動する。特定のコマンド・アドオンをロードするには、コマンド・ランタイムの起動時に、システムプロパティ system.properties.xml に対して、ロードしたいコマンド・アドオンが記述されたプロパティファイルを設定する必要がある。以下に、コマンド・アドオンのロード方法を示す。

```
set DARUMA_HOME=c:\darumatool
java -Dapplication.home="%DARUMA_HOME:\=/"
    -Dsystem.properties.xml="./etc/properties.xml" -jar startup.jar
```

コマンドを実行するカレントディレクトリが本ツールのホームである場合は、-Dapplication.home は省略可能である。

2.4.3 コマンド・アドオンの設定ファイル

コマンドアドオンの設定ファイルは、etc ディレクトリの各コマンド名のディレクトリ以下に格納されている。設定ファイルには、system.properties.xml と system.context.xml の2種類の XML ファイルがある。

system.properties.xml ファイル system.properties.xml は、コマンド・ランタイム及びコマンド・アドオンの動作を設定するものであり、以下の system.context.xml のパスや、起動するメインクラスを記述する。サーバーモードの切り替えもこのファイルを行うことができる。メインクラスを変更することによって、様々なコマンド・アドオンを同一のコマンド・ランタイム上で起動することができる。以下に system.properties.xml の設定例を示す。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<!--モード切替 -->
<entry key="system.server.mode">false</entry>
<entry key="system.silent.mode">true</entry>
<!-- システムパラメータ -->
<entry key="system.context.cfg.xml">
${application.home}/etc/darumaclient/system.context.xml
</entry>
<!-- ランタイムパラメータ -->
<entry key="application.name">DarumaClient for MISP</entry>
<entry key="application.description">
2006 AIST DaRuMa Project - Daruma Client for MISP
</entry>
<entry key="application.main.class">daruma.main.DarumaClient</entry>
<!-- アドオンパラメータ -->
<entry key="mispssoapclient.xslt.copy.template">
${application.home}/work/xsl/standard_copy.xslt
</entry>
<entry key="darumaclient.retrieve.soap.body.xslt.template">
${application.home}/work/xsl/retrieve_soap_body.xslt
</entry>
</properties>
```

system.context.xml ファイル system.context.xml は、DI コンテナで処理される Bean 定義ファイルである。本ツールでは、コマンドラインのオプション及びパラメータの定義に使用している。以下に system.context.xml の設定例を示す。


```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">

<beans>
<bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
</bean>
<bean id="commandLineHelper" class="application.runtime.CommandLineHandler">
<property name="commandName"
value="ShowFeatureTypes [-debug] [-out OUTPUT_FILE] -host HOSTNAME"/>
<property name="parser">
<bean class="org.apache.commons.cli.BasicParser"/>
</property>
<property name="optionList">
<list>
<bean class="application.runtime.CommandLineOption">
<property name="optionName" value="debug"/>
<property name="description" value="中間ファイルを削除しません。"/>
<property name="hasValue" value="false"/>
<property name="required" value="false"/>
<property name="argName" value=""/>
<property name="argCount" value="0"/>
<property name="optionalArg" value="false"/>
</bean>
<!-- Server Parameters -->
<bean class="application.runtime.CommandLineOption">
<property name="optionName" value="host"/>
<property name="description" value="サーバのホスト名又は IP アドレス"/>
<property name="hasValue" value="true"/>
<property name="required" value="true"/>
<property name="argName" value="HOSTNAME"/>
<property name="argCount" value="1"/>
<property name="optionalArg" value="false"/>
</bean>
<bean class="application.runtime.CommandLineOption">
<property name="optionName" value="out"/>
<property name="description" value="出力ファイル名 (XML ファイル)"/>
<property name="hasValue" value="true"/>
<property name="required" value="false"/>
<property name="argName" value="OUTPUT_FILE"/>
<property name="argCount" value="1"/>
<property name="optionalArg" value="false"/>
</bean>
</list>
</property>
</bean>
</beans>

```

2.5 システムプロパティによるパラメータ設定

各 Processor オブジェクトやコマンドは、システムプロパティによって、動作が決定される。システムプロパティを設定する方法は、コマンド・ランタイム及びコマンド・アドオンの双方で用意されており、適用される優先度によって使い分けることができる。

1. Java VM の-D オプションによる設定値
2. etc ディレクトリ以下の default.system.properties.xml ファイルの値
3. etc ディレクトリ以下の各コマンド用ディレクトリの system.properties.xml ファイルの値
4. work/sysprops ディレクトリ以下の拡張子が xml であるファイルの値
5. プログラム内で System オブジェクトに対して設定した値