Semantix
INFORMATION TECHNOLOGIES

Roaming Components User Manual

revision: 1.9
December 2006

Contents

List of Figures

Chapter 1

Introduction

Semantix Roaming Components is a collection of versatile command line applications that operate on TAP3 files in an autonomous manner; without any human intervention.

Roaming Components adopt a UNIX-like tools philosophy whereby complex tasks are performed by wiring together a collection of simpler, standardized, independently tested and evolved modules or tools. This is contrasted with the philosophy of providing monolithic applications that supposedly solve all your problems (when in fact, due to their complexity, introduce major problems of their own).

The following technical principles have guided the architecture and implementation of the Semantix roaming components:

Modularity: Semantix roaming components were specifically designed with modularity in mind. The provided suite is not a single, tightly-knit application (though the end-user can always consider it as a single application from his point of view). Instead of shoe-horning every conceivable requirement into a one-size-fits-everything application, Semantix has developed a suite of self-contained modules (components), each of which can be used as an independent application on its own. By standardizing the way all these modules interact with their environment they can be easily combined together to provide more powerful solutions.

Technology: *Semantix Roaming Components* are based on highly-optimized C++ code for the hard tasks of TAP3 processing, like validations, conversions, et.c. C++ is unrivalled when it comes to performance, maturity and making the best use of available resources. C++ compiler technologies have been continuously honed for the past 20 years for a vast array of hardware / OS platforms and overall provide the best solution when it comes to low-level custom development.

Environment-neutral: From a programming environment perspective, the modules implemented in C++ make as few assumptions about the environment in which they operate as possible (close to none). The C++ implementation is "pure": it is only concerned with the abstract logic and the algorithms necessary to process TAP files. In fact the entire C++ implementation uses only the standard language libraries so that the full source code can be compiled and built in every system that features a decent C++ compiler/linker.

No external libraries: The entire system is built using code that Semantix owns. Semantix has developed its own ASN.1-compiler to be totally independent of ASN.1 compiler vendor's software.

Configurable: Many components are configurable via XML files. Since these files contain simple XML constructs, they can be directly created, edited or adapted to new requirements. In many cases, there is a default configuration "inside" a component, allowing it to work without an external XML configuration file (e.g. see section 2.3, the TAP_Validator).

Standards-based: The entire implementation is based on published, open standards and technologies. No closed source third-party tools or libraries are utilized.

Chapter 2

Roaming Components

2.1 *ASCII2TAP*

The *ASCII2TAP* module is used to create a TAP file out of an incoming ASCII file.

The inputs / outputs of the *ASCII2TAP* module are depicted in Figure 2.1.

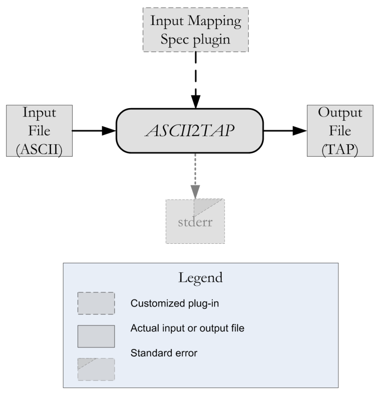

Figure 2.1: *ASCII2TAP inputs and outputs*

The *ASCII2TAP* module is typically triggered with the following command line:

```
ASCII2TAP [-v] [-V] [-h] [-fixup]
          -ims plugin.dll
          -i inputTextFileName -o createdTapFileName
```

Square brackets indicate optional arguments. Arguments are thus:

- "`-v`" requests verbose mode (additional 'v's as in "`-vv`" or "`-vvv`" increase verbosity)
- "`-V`" prints the component's version number
- "`-h`" prints a summary of the component's command line options
- "`-fixup`" requests an update of the AuditControlInfo structure at the end of the mapping. This guarantees that the generated TAP file's "summary" section contains up-to-date information on the number of calls, the total charges, et.c.
- "`-ims`" points to the input plugin that provides the mapping specification between the incoming ASCII file and the outgoing TAP file. The TAP version of the TAP file to be created is specified inside the plugin's code.
- "`-i`" points to the incoming ASCII file
- "`-o`" points to the outgoing TAP file (to be created)

Upon invocation, the *ASCII2TAP* module converts the incoming ASCII file into the specified TAP file, mapping the fields as dictated by the code in the plugin. Notice that since the plugin contains compiled C code[1] it can perform arbitrarily complex mapping rules — calculated fields, etc.

The *ASCII2TAP* roaming component currently supports the following TAP formats: 3.1, 3.2, 3.3, 3.4, 3.9. 3.10 and 3.11.

[1]For more on creation of plugins, see the whitepaper on *ASCII2TAP* available from `http://www.tapeditor.com`.

2.2 *TAP2ASCII*

The inputs / outputs of the *TAP2ASCII* module are depicted in Figure 2.2.

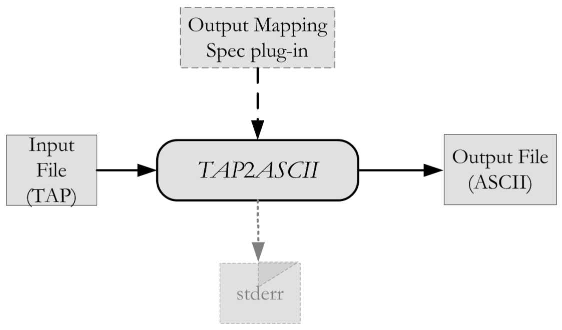

Figure 2.2: *TAP2ASCII inputs and outputs (for legend see Figure 2.1).*

The *TAP2ASCII* module is typically triggered with the following command line (expanded over more than one line for ease of reference):

```
TAP2ASCII [-v] [-V] [-h]
          -ims plugin.dll
          -i inputTapFileName -o createdTextFileName
```

Square brackets indicate optional arguments. Arguments are thus:

- "`-v`", "`-V` and "`-h`": same as in §2.1
- "`-ims`" points to the plugin used to perform the mapping of TAP information to a specific ASCII format. Notice that the plugin code contains all the mapping logic - i.e. where to read the information from, as well as how to output it.
- "`-i`" points to the incoming TAP file
- "`-o`" points to the outgoing ASCII file (to be created)

Upon invocation, the *TAP2ASCII* module converts the incoming TAP file into the specified ASCII format, mapping the fields appopriately by executing the code inside the plugin. Since the plugin contains compiled C code[2], arbitrarily complex mappings can be performed, i.e. calculated fields et.c.

The *TAP2ASCII* roaming component currently supports the following TAP formats: 3.1, 3.2, 3.3, 3.4, 3.9. 3.10 and 3.11.

2.3 *TAP_Validator*

The *TAP_Validator* module is used to validate an incoming TAP file against the applicable TD.57 validations. It also supports operator-specific selection of validation sets and validation parameters.

The inputs / outputs to the *TAP_Validator* module are depicted in Figure 2.3.

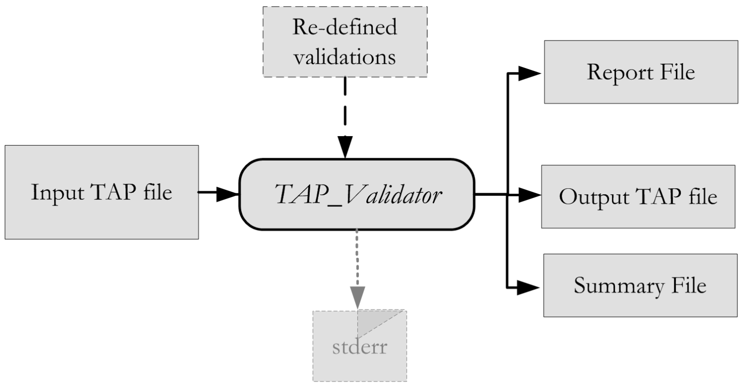

Figure 2.3: *TAP_Validator inputs and outputs (for legend see Figure 2.1).*

The *TAP_Validator* module is triggered with the following command line:

```
TAP_Validator [-v] [-V] [-h]
              [-o createdTapFileName]
              [-report reportFile]
              [-iv redefinedValidations.xml]
              [-s summaryFile]
```

[2]For more on creation of plugins, see the whitepaper on *TAP2ASCII* available from `http://www.tapeditor.com`.

```
                [-rtf rtfFile]
                [-o createdTapFileName]
                -i inputTapFileName
```

Square brackets indicate optional arguments. Arguments are thus:

- "`-v`", "`-V` and "`-h`": same as in §2.1
- "`-i`" points to the incoming TAP file to be validated
- "`-o`" points to the outgoing TAP file (incoming file minus offending (severe) calls (in this way, you can easily remove calls that trigger severe errors from a file) [3].
- "`-report`" points to an extensive text report to be created, detailing all warnings and severe/fatal errors with path information - depicting exactly where they occured in the TAP tree.
- "`-iv`" points to an XML file that redefines the validation ruleset to apply as well as the values to use for the validations (i.e. Service Level Agreement (SLA) parameters).
- "`-s`" points to a summary text report to be created, detailing the categories of warnings and severe/fatal errors that occured in the file.
- "`-rtf`" just like "`-s`", but generates an RTF report (for easy printing and reporting to e.g. the revenue assurance department).

Through the configuration XML file ("`-iv`" option), the component allows complete customization of the validations logic on a per-roaming-agreement basis. What this means is that the same TD.57 validations may be applied very differently for each one among the hundrends of the roaming agreements an operator may have in place. These are called validation specializations and are stored in configuration XML files - one per agreement.

The following types of validation specializations are supported:

[3]. . . in the case where only severe errors are found. If fatal errors are also found then clearly no outgoing TAP file can be created.

- adjust various ranges and value sets used by a validation.
- adjust the gravity of specific validations for specific roaming agreements. Declare for instance that a specific validation for a specific roaming agreement should be treated as a warning instead of as a severe error or declare that a specific fatal-level validation should be disabled for a specific roaming agreement.
- adjust values and toggle options for “billaterally agreed” options.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<TapValidator>
    <TapVersion mjVersion="3" mnVersion="11">
        <Sets>
          <set id="SupplServiceActionCodeSSSet" type="int">
                <value description="Registration">0</value>
                <value description="Erasure">1</value>
                <value description="Activation">2</value>
                <value description="Deactivation">3</value>
                <value description="Interrogation">4</value>
                <value description="Invocation">5</value>
                <value description="Reg. of Password">6</value>
          </set>
          <set id="SupplServiceActionCodeUSSDSet" type="int">
                ...
```

The listing above is an example of the first category: value sets are defined here, and in particular acceptable values for the “`SupplServiceActionCodeSSSet`” are defined. Notice that the value sets are TAP version specific (in this case, the rule set is defined for the respective TAP 3.11 validation - see mnVersion attribute). This valueset is referenced further down in the XML file, in the “`Validations`” section:

```
        ...
        </Sets>
        <Validations>
          ...
          <validation element="Action Code" errorCode="20"
               context="SS" enabled="true" severity="S">
            <setsUsed>
              <setRef>SupplServiceActionCodeSSSet</setRef>
            </setsUsed>
            <BillAgreements>
            </BillAgreements>
          </validation>
            ...
```

This is the second category of configurability: it allows the user of *TAP_Validator* to select:

- Whether the validation is enabled or not, through the "`enabled`" attribute.
- Whether the validation triggers a "`W`"arning, "`S`"evere or "`F`"atal error, through the "`severity`" attribute.

```
  ...
  <BillateralAgreements>
    <entry id="strAccountingInfo_TapCurrency" value="SDR" />
    <entry id="bUseOfContentTransactionAgreed" value="true" />
    <entry id="bUseOfServiceCentreUsageAgreed" value="true" />
     ...
```

Finally, this is a section of the third category of configurability: it allows modification of values that influence specific TD57 validations - those that refer to "billaterally agreed" data.

The fact that validation specializations are stored as XML files allows for easy and future-proof management of all SLAs. Editing this XML configuration file (even through a simple editor like VI, since the structure is quite simple) one can easily direct the validator to accomodate complex validation requirements. The XML file used is depicted as "Re-defined validations" in the diagram of Figure 2.3, and defines all validations applicable to a particular roaming agreement (including validation specializations that override default behaviour).

If no XML file is specific, default values are used for all validations (as set inside the TD.57 standard).

The *TAP_Validator* roaming component currently supports the following TAP formats: 3.1, 3.2, 3.3, 3.4, 3.9. 3.10 and 3.11.

2.4 $TAP_x 2 TAP_y$

The $TAP_x 2 TAP_y$ module is used to convert an incoming TAP file of one version into a TAP file of another version.

The inputs / outputs to the $TAP_x 2 TAP_y$ module are depicted in Figure 2.4.

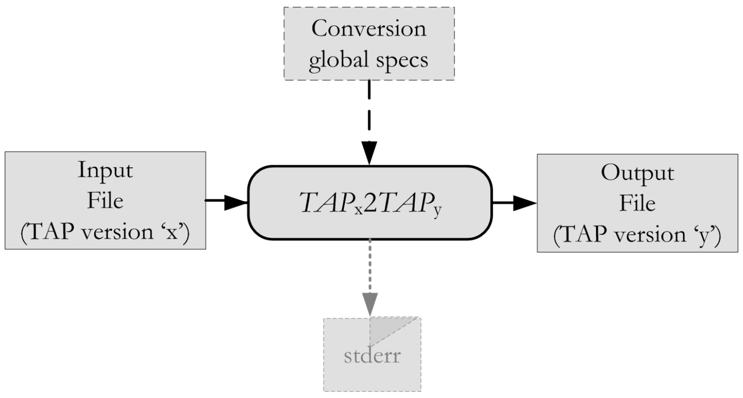

Figure 2.4: *$TAP_x 2 TAP_y$ inputs and outputs (for legend see Figure 2.1).*

The $TAP_x 2 TAP_y$ module is typically triggered with the following command line:

```
TAPx2TAPy [-v] [-V] [-h]
          [-globals configurationFile.xml]
          [-fixup]
          -i inputTAPxFileName
          -o createdTAPyFileName
          -ov outputVersion
```

Square brackets indicate optional arguments. Arguments are thus:

- "`-v`", "`-V` and "`-h`": same as in §2.1
- "`-i`" points to the incoming "TAP$_x$" file
- "`-ov`" specifies the TAP version of the outgoing TAP file (the TAP version of the incoming TAP file is determined automatically)
- "`-o`" points to the outgoing "TAP$_y$" file to be created
- "`-fixup`" requests an update of the AuditControlInfo structure at the end of the conversion. This guarantees that the generated TAP file's "summary" section contains up-to-date information on the number of calls, the total charges, et.c.
- "`-globals`" points to an XML file that defines conversion parameters.

The conversion parameters XML file contains two sections, "Countries" and "PMNs":

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<TAPx2TAPy version="1" >
    <Countries>
        <CountryIdToCode CountryId="AFG" DialingCode="93"  />
        <CountryIdToCode CountryId="ALB" DialingCode="355" />
        <CountryIdToCode CountryId="DZA" DialingCode="213" />
         ...
    </Countries>
    <PMNs>
        <PMN PMNID="BRAC3" DialingCode="55" />
        <PMN PMNID="BRAC4" DialingCode="55" />
        <PMN PMNID="BGRVA" DialingCode="359" />
         ...
    </PMNs>
</TAPx2TAPy>
```

As seen in the above sample, these two sections provide the mappings between...

- CountryIDs and DialingCodes
- PMNIDs and DialingCodes

The TAP_x2TAP_y roaming component includes a static copy of these values that covers a large percentage of the active operators at this time. If however the need arises to convert a file that comes from an unknown operator, this XML file allows the user of TAP_x2TAP_y to add new values. The component comes with a default file, which can be extended with new entries at will.

The TAP_x2TAP_y roaming component currently supports the following TAP formats: 3.1, 3.2, 3.3, 3.4, 3.9. 3.10 and 3.11.

2.5 *TAP_Splitter*

The *TAP_Splitter* module is used to split the calls inside an incoming TAP file into those that obey a "splitting specification" and those that do not. Those that do, are stored in a new TAP file.

The inputs / outputs to the *TAP_Splitter* module are depicted in Figure 2.5.

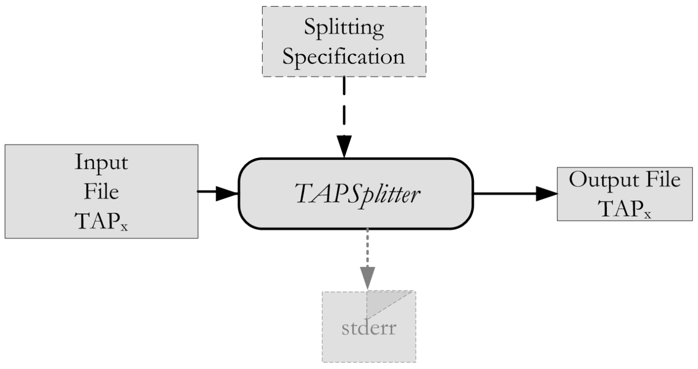

Figure 2.5: *TAP_Splitter inputs and outputs (for legend see Figure 2.1).*

The *TAP_Splitter* module is triggered with the following command line (expanded over more than one line for ease of reference):

```
TAP_Splitter [-v] [-V] [-h]
             [-fixup]
             [-negate]
```

```
              −f callSplittingSpefication.xml
              −i inputTapFileName
              −o createdTapFileName
```

Square brackets indicate optional arguments. Arguments are thus:

- “-v”, “-V and “-h”: same as in §2.1
- “-f”: points to the XML file containing the “splitting specification”. A splitting specification file typically defines an arbitrary condition (boolean expression) - a filter - for the incoming file. Each condition is evaluated for each call and if the call satisfies it, the outgoing file will include a copy of the call.
- “-o”: points to the file to be created.
- “-fixup”: requests an update of the AuditControlInfo structure at the end of the filtering. This guarantees that the generated TAP file’s “summary” section contains up-to-date information on the number of calls, the total charges, et.c.
- “-negate”: calls that don’t obey the splitting specification are those that are included in the output file.

The “splitting specification” controls which calls will be included in the output file.

```
<CallSearchData AndOrOr="AND" FromCallIndex="0"
      ToCallIndex="500">
  <!-- CallTypes section -->
  <CallType value="MobileOriginatedCall" />
  <CallType value="MobileTerminatedCall" />
  ...
  <!-- Additional rules section -->
  <CallSearchSingleRule FieldTypeName="Charge"
    FieldVarName="charge" Condition="EQ" Value="10" />
  <CallSearchSingleRule FieldTypeName="ChargedPartyLocation"
    FieldVarName="" Condition="EXISTS" Value="" />
  ...
</CallSearchData>
```

As seen in the sample above, the splitting specification XML file, contains two sections:

- The CallTypes section provides an easy way to select calls based on their type (e.g. MobileOriginatedCall, GprsCall, etc).
- The Additional rules section allows the user of *TAP_Splitter* to specify extra conditions (boolean expressions) that should be met on the selected calls. In the example above, MOC and MTC calls that contain at least one "Charge" with a value of 10, AND have a ChargedPartyLocation field present are output. The "AND" comes from the "`AndOrOr`" attribute of "`CallSearchData`, and could also be "OR". Additionally, the FromCall-Index and ToCallIndex attributes allow filtering based on call indices.

The *TAP_Splitter* roaming component currently supports the following TAP formats: 3.1, 3.2, 3.3, 3.4, 3.9. 3.10 and 3.11.

2.6 *TAP_Merger*

The *TAP_Merger* module is used to merge a number of incoming TAP files (of the same TAP version) into a single TAP file (of the same TAP version).

The inputs / outputs to the *TAP_Merger* module are depicted in Figure 2.6.

The *TAP_Merger* module is triggered with the following command line (expanded over more than one line for ease of reference):

```
TAP_Merger [-v] [-V] [-h]
        -i inputFile1 inputFile2 [inputFile3] ...
        -o createdTapFileName1
```

Square brackets indicate optional arguments. Arguments are thus:

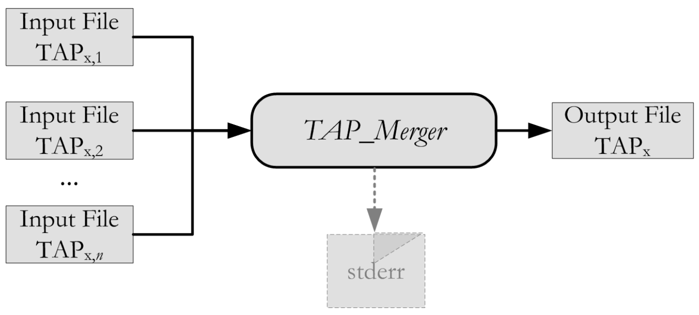

Figure 2.6: *TAP_Merger inputs and outputs (for legend see Figure 2.1).*

- "`-v`", "`-V` and "`-h`": same as in §2.1
- "`-i`" points to the list of input TAP files (at least two must be specified)
- "`-o`" points to the merged TAP file to be created

The *TAP_Merger* roaming component currently supports the following TAP formats: 3.1, 3.2, 3.3, 3.4, 3.9. 3.10 and 3.11.

2.7 *TAP_Anonymizer*

The *TAP_Anonymizer* module removes "sensitive" information from a TAP file according to an "anonymization specification".

The inputs / outputs to the *TAP_Anonymizer* module are depicted in Figure 2.7.

The *TAP_Anonymizer* module is triggered with the following command line (expanded over more than one line for ease of reference):

```
TAP_Anonymizer [-v] [-V] [-h]
        -i inputTapFileName
        -o createdTapFileName
        -fs anonymizationSpecification.xml
```

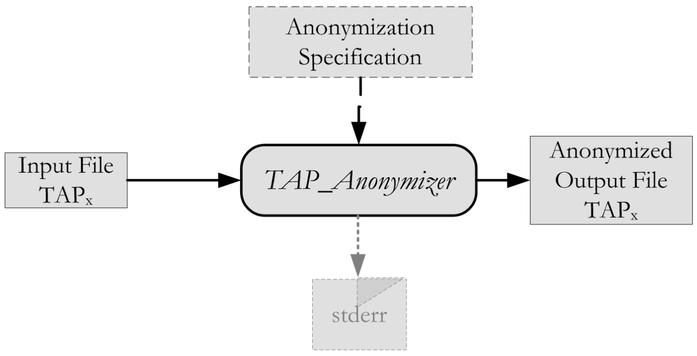

Figure 2.7: *TAP_Anonymizer inputs and outputs (for legend see Figure 2.1).*

Square brackets indicate optional arguments. Arguments are thus:

- "`-v`", "`-V` and "`-h`": same as in §2.1
- "`-fs`" points to the XML file defining the anonymization logic. This file defines what fields are construed to contain "sensitive" information and provides actions to replace those sensitive values.
- "`-i`" points to the incoming TAP file to be anonymized
- "`-o`" points to the anonymized TAP file to be created

```
<TapAnonymizer>
    <Modify nodeType="Msisdn" action="constant"
        value="306941234567" />
    <Modify nodeType="Charge" action="random"
        minValue="0" maxValue="256" />
</TapAnonymizer>
```

The above anonymization specification XML file:

- replaces all Msisdns with the value 306941234567
- replaces all Charge nodes with random values that range between 0 and 256 (included)

The *TAP_Anonymizer* component can be used whenever the need arises for sharing of TAP files with a partner (say, a roaming partner) and the legal rules applicable in your country prohibit sharing call information. Through the configuration XML file, you can specify arbitrarily complex anonymization rules.

The *TAP_Anonymizer* roaming component currently supports the following TAP formats: 3.1, 3.2, 3.3, 3.4, 3.9. 3.10 and 3.11.

2.8 *TAP2XML*

The *TAP2XML* component is used to create XML files from TAP3 input files (presumably received by another PMN or generated by other systems).

The inputs / outputs to the *TAP2XML* module are depicted in Figure 2.8.

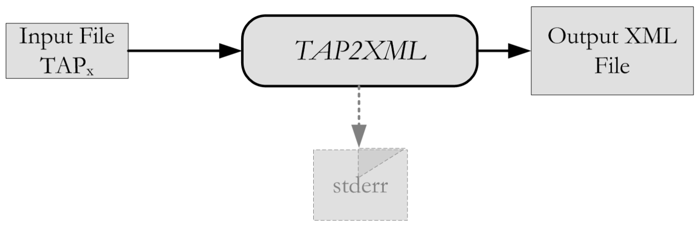

Figure 2.8: *TAP2XML inputs and outputs (for legend see Figure 2.1).*

The *TAP2XML* module is triggered with the following command line (expanded over more than one line for ease of reference):

```
TAP2XML [-V] [-h]
        [-o createdXmlFileName]
        -i inputTapFileName
```

Square brackets indicate optional arguments. Arguments are thus:

- "`-V` and "`-h`": same as in §2.1
- "`-i`" points to the input TAP file
- "`-o`" points to the XML file to be created (if argument is missing, output goes to `stdout`

An example output XML file is shown below:

```
<?xml version="1.0" encoding="utf-8" ?>
<TAP3File mjVersion="3" mnVersion="10"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
     <DataInterChange>
          <transferBatch>
               <batchControlInfo>
                    <sender>DEUD2</sender>
                    <recipient>GRCPF</recipient>
                    <fileSequenceNumber>10000</fileSequenceNumber>
                    <fileCreationTimeStamp>
                    ...
```

The *TAP2XML* roaming component currently supports the following TAP formats: 3.1, 3.2, 3.3, 3.4, 3.9. 3.10 and 3.11.

2.9 *XML2TAP*

The *XML2TAP* component is used to create TAP files from XML input files (presumably generated by TAP2XML).

The inputs / outputs to the *XML2TAP* module are depicted in Figure 2.9.

The *XML2TAP* module is triggered with the following command line (expanded over more than one line for ease of reference):

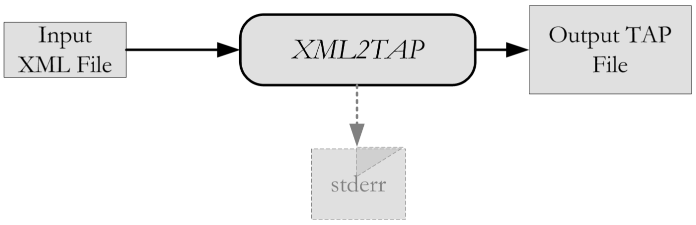

Figure 2.9: *XML2TAP inputs and outputs (for legend see Figure 2.1).*

```
XML2TAP [-V] [-h]
        [-o createdTapFileName]
        -i inputXmlFileName
```

Square brackets indicate optional arguments. Arguments are thus:

- "`-V` and "`-h`": same as in §2.1
- "`-i`" points to the input XML file
- "`-o`" points to the TAP file to be created

The *XML2TAP* roaming component currently supports the following TAP formats: 3.1, 3.2, 3.3, 3.4, 3.9. 3.10 and 3.11.

Chapter 3

Supported platforms

The *Semantix Roaming Components* are built using ISO standard C++ code (ISO C99 C++[1]) and can be compiled and installed in every system featuring a modern C++ compiler (compiled binaries are also offered for some platforms — see below).

The above “requirements” are such that are met by practically every modern platform and are certainly met by the following platforms:

- Commercial systems
  - Solaris 8 and later (SunOS 5.8 and later, both SPARC and x86 architectures)
  - HP-UX 11i v1 and later
  - IBM AIX 5L or Linux
  - Microsoft Windows NT and later
  - Any UNIX system with an ISO C99 C++ compiler
- Open source kernels (any supported CPU - Intel, AMD, PowerPC, ...)

[1]By this time, practically all modern C++ compilers for every platform support ISO C99 C++.

  - Linux (any distribution - SuSE, Debian, RedHat, Madrake - with any kernel from 2.4.3 - 2.6.20)
  - FreeBSD (4.8 and later)
  - OpenBSD (3.5 and later)
  - NetBSD (2.0 and later)

Precompiled binaries are provided and directly supported if the destination platform is one of the following:

- Solaris (sparc, x86)
- Linux (x86, amd64)
- FreeBSD (x86)
- OpenBSD (x86)
- Windows 2000, Windows XP, Windows Vista