DELFT-FEWS

Published Interface

Version 2.5

21 March 2005

CLIENT:	
TITLE:	Published Interface
REFERENCES:	

VER.	ORIGINATOR		DATE	REMARKS	REVIEW		APPROVED BY	
0.9	Schellekens		December 9,	draft	Werner		Heynert	
1.0	Schellekens		January 23, 2003	final draft	Werner		Heynert	
2.0	Werner		February 20, 2003	final	Heynert		Bons	
2.1	Schellekens/vdAkker		July 10, 2003	final	Werner		Segeren	
2.2	Heynert		September 10, 2003	update	Werner		Ruijgh	
2.3	Heynert		September 24, 2003	update	Heynert		Akker	
2.4	Schellekens/vanKappel		November 25 2004	update	Werner		Ruijgh	
2.5	Schellekens		March 21, 2005	update	Werner		Ruigh	

PROJECT IDENTIFICATION:	Q3049
KEYWORDS:	DELFT FEWS, NFFS, Flow Forecasting System, Published Interface, General Adapter
NUMBER OF PAGES	
CONFIDENTIAL:	☐ YES, until (date) ☐ NO
STATUS:	☐ PRELIMINARY ☐ DRAFT [✓ FINAL]

Guide to the reader

This document describes the interface (data types and formats) that modules need to adhere to when they are to work within the DELFT-FEWS system, together these form the DELFT-FEWS published interface. The document describes the recognised file formats. This includes an extensive description of the formats as defined in XML-Schema files. These are included in the Appendices.

New in version 2.1 (compared to 2.0)

The original General Adapter Specifications document has now been split in two parts. One part describing the General Adapter configuration and one part (this part) describing the files and formats that constitute the published interface. This information is essential to third parties undertaking development of module adapters to interface with NFFS. To avoid confusion the documentation required for configuring the general adapter, itself a part of NFFS, has been moved to the other part. The information in this part should be sufficient for the construction of module adapters for the system. In the new version of the published interface all the separate schema's have been incorporated into a single schema file: fileformats.xsd. Most of the root elements are the same as those in the previous version (see below for a summary of the changes). Apart from the changes to the root elements described below, xml files that conform to the previous implementation can be left unaltered apart from the schema location which is now fileformats.xsd for all types. The following changes have been made since the previous version of the file formats specifications:

- *All:* A new TimeStep type has been introduced. This affects the following root elements/file formats : TimeSeries, MapStacks, Longitudional profiles, Polygons.
- *All:* all root elements now have a required version attribute that must be set to 1.1 (was 1.0)
- *All:* all root elements (file formats) that have a date/time child now have an (optional) timeZone element.
- *All:* A new name conventions is used for elements
  - *All root elements and complex types start with a upper case letter*
  - *If an element (or an attribute) name is composed of more than one word each word (except the first) is capitalized.*
- *Time series*: the ascii file format is no longer supported in the published interface.
- *Time series*: the schema definition now allows for multiple series in a single file
- *Time series*: the timeseriesId element is removed. Instead the locationId element is now required and a parameter element is introduced. The content element is removed.
- *Time series*: the startDate and endDate elements are now required.
- *Locations*: The model of the children of the location element has been changed from all to sequence (for most practical purposes the entails no change since the previous version)
- *Map stacks:* the schema definition now allows for multiple series in a single file.
- *Map stacks*: attributes of the mapStack element (TimeStep, startDate, endDate) have been converted to (required) child elements. The version attribute has been moved to the new root element (MapStacks)
- *Map stacks*: The gridid element has been removed, a (required) locationId element has been added, the parameterId element has been renamed to parameter

- *Map stacks:* the event element has been added. This is used for specifying the date and time with non-equidistant map stacks.
- *Grid cell centre points:* the name of the root elements has been changed from locations to Cells.
- *Module state data:* a stateId element (required, defined by the module) has been added. The (optional) stateName, and dateTime elements have been added.
- *Longitudinal profile:* the schema definition now allows for multiple series in a single file.
- *Longitudinal profile:* locationId ,parameter, startDate, endDate, TimeStep and (optional) branchId elements have been added.
- *Longitudinal profile:* the profile element holds a new event element. The xData element used to be a child of the profile element, it is now a child of the event element.
- *Module parameters:* the optional moduleId element has been removed.
- *Lookup tables:* the root element has been renamed to table. The structure of the root element has changed considerably and now allows for an unlimited number of columns.
- *Module diagnostics output:* the level order has been reversed (see text). No changes to elements or attributes.
- *Branch data:* the attributes of the branch element (-branchID, branchName, startChainage, endChainage, upNode, downNode) have been converted to child elements.
- *Cross-sections:* all the attributes of the crossSection element have been converted to child elements. XsectID has been renamed to xSectionId.
- *Lateral inputs:* the link to timeseries date for the lateral input is now performed using the locationId/parameter combination. The timeseriesid and the timeseriesfile elements are removed.
- *Polygons*: the schema definition now allows for multiple series in a single file.
- *Polygons*: the points element is changed and is now unbounded to allow for time series.
- *Polygons*: the polygonid elements is removed, the locationId and parameter elements have been added.

Grid cell centre points were grouped among the dynamic data types, they are now classified as static data. Conversely, polygons are now listed as dynamic data.

A graphical overviews of the root elements has been added to the description of each format. The complete schema documentation remains in appendix D.

New in version 2.2

Chapter 3 has been updated.

In Chapter 4 the application of module adapters has been added which was also included version 2.0 of this document but has fallen out in version 2.1.

Appendix F is added describing in more detail the interaction between the NFFS and external modules.

New in version 2.3

The updated version 2.3 was made after discussion with HR Wallingford and CEH Wallingford regarding their module adapters.

- The version number of the root elements has been changed to 1.2
- Although the schema does allow otherwise, it has been decided that all datapoints in the timeseries files should be in chronological order.
- A typing error (devider -> divider) has been corrected in the TimStep type
- The single schema (fileformats.xsd) has been split into several files, each containing a single file-format (root element).

New in version 2.4

- Document has been updated to reflect that the PI is a part of DELFT-FEWS and not the NFFS only.

New in version 2.5

- Added memo in appendix F
- Updated explanation in states and diagnostics sections
- Added a questions and answers section

Contents

1 Introduction ..... 1—1
2 Testing ..... 2—1
3 Relation with other XML standards ..... 3—1
4 Running modules from Delft-FEWS ..... 4—1
4.1 Principles ..... 4—1
4.2 Module runtime environment ..... 4—2
4.3 Data exchange between General Adapter and Modules ..... 4—3
5 General Remarks ..... 5—1
6 Introduction to File Format Specifications ..... 6—1
7 Dynamic data ..... 7—1
7.1 Time Series ..... 7—1
7.2 (Time series) Location ..... 7—4
7.3 Map (stacks) data (grids) ..... 7—5
7.4 Module state data ..... 7—7
7.5 Longitudinal profile ..... 7—9
7.6 Module parameters ..... 7—11
7.7 Lookup tables ..... 7—12
7.8 Module diagnostics output ..... 7—14
7.9 Polygons ..... 7—15
8 Static data ..... 8—1
8.1 Grid cell center points ..... 8—1
8.2 Branch data ..... 8—2
8.3 Cross-sections ..... 8—4
9 Lateral inputs ..... 9—1
10 List of files ..... 10—1

List of Appendices

A XML Schema listings A–1

B Grid data file formats B–1

B.1 PCRaster B–1

B.2 USGS BIL / BSQ / BIP B–1

B.3 Image file B–2

B.4 ESRI ASCII B–2

C EA Schema to PI mapping C–1

D Quality flags D–1

E Schema Documentation E–1

F Module Adapter Application F–1

G Questions and aswers G–1

1 Introduction

This document describes the interface (data types and formats) that modules need to adhere to when they are to work within the DELFT-FEWS system, together these form the DELFT-FEWS published interface. This document describes the recognised file types.

All data exchange is done by means of Extended Markup Language (XML) files. XML is a markup language used to describe the structure of data in a meaningful way. XML is widely used and as it is an open standard many free (and commercial) tools exists to create XML documents.

XML Schemas are used to describe the contents of XML documents. An XML schema (see `http://www.w3.org/XML/Schema` for more information) is available for each file type. The exception in the interface published here being gridded data that is stored in separate (binary or ASCII) files and (optionally) ASCII based time series that can be used by the general model adapter.

Although the description of the file formats can be extracted from the XML schema files, Appendix E includes a tabular description of all the Schema (XSD) files. In addition, XML sample documents are provided for all file formats. However, if the schema documentation is not complete (or clear), the actual schema (XSD) files should be consulted. To keep the document to a manageable size, these files have not been printed and included in the documentation. The files are available in digital format.

2 Testing

WL | Delft Hydraulics maintains a number of XML schema files that describe the layout of the files that together form the NFFS published interface. To validate if XML files adhere to the interface, the schema files can be used in combination with one of available schema validators. The w3 consortium maintains a list at:

```
http://www.w3.org/XML/Schema
```

The example files in this document have been validated using the jedit XML mode (available freely at www.jedit.org) and by using XMLSPY, but many other (free and commercial) alternatives are available.

The ASCII and binary grid files should be prepared according to the specifications given in Appendix B.

3 Relation with other XML standards

Similar to the establishment of an XML based published interface between third party modules and DELFT_FEWS through the General Adapter, the U.K. Environment Agency has established XML schemas as a standard for exchange of data between the various sub-systems of the Agency. This is particularly relevant to the Flow Forecasting Systems under development as it will include the exchange of data between this system and external data sources (RTS, DDS etc.).

The approach taken was compared to that used in the published interface described in this document. Although the two definitions are strictly separate, where the published interface described here is used to communicate between the DELFT-FEWS system and third party modules, while the EA Schemes are used for exchange of data between the different systems within the EA, it is seen as a potential advantage that the different schemes are comparable. It is accepted that the schemes may differ.

In the comparison of the schemes attention has only been paid to the Schema established for hydrometric data and the time series definition used here. The differences between the two schemas have been reviewed and where appropriate amendments have been made to the published interface time series format such that relevant information can be easily mapped from one scheme to the other. In the appendices of this document a spreadsheet is supplied indicating this mapping.

The main differences between the DELFT-FEWS Published Interface and the EA XML schemes comprise:

- The EA Schema contains more extended header information. Where appropriate these fields have been included in the published interface time series formats. Although the available fields are extensive, most are optional and need not be supplied if not available.
- The geographic datum included in the EA scheme considers only the British National Grid. In the published interface different datums may be considered and must be explicitly named.
- Both the published interface and the EA hydrometric data schemes apply a philosophy of a single data stream for each data type. This is in contrast with the tidal scheme where multiple data types are contained in each data stream. This approach is not advocated as this may endanger the openness of the system.

4 Running modules from Delft-FEWS

In this section the interaction between the DELFT-FEWS and external modules is described. Additional details are given in Appendix F.

4.1 Principles

To allow running modules within the DELFT-FEWS through the general adapter, these modules must adhere to a number of conventions. The general adapter does provide a flexible interface, which entails that there is not a singular strict definition on how modules are run. However, the options provided do not allow complete freedom. This section describes in detail how a module is run and what conventions exist.

In all cases, the central philosophy of the general adapter is that it is to as large an extent as possible ignorant of module specific details. This intelligence is strictly separated from the DELFT-FEWS side of the system, as this guarantees an open system. Module specific intelligence required by the module for running this in real time is vested in the module adapter itself. Figure 1 gives a schematic view on the interaction between DELFT-FEWS (central database) on the one side and the Module on the other. Communication is established through the published interface. The general adapter (this module is a part of DELFT-FEWS) is configured to provide the required data (both dynamic and static) in the published interface format. A module adapter (supplied by the same supplier as the module itself) is used to translate the data from the published interface format to the native module format.

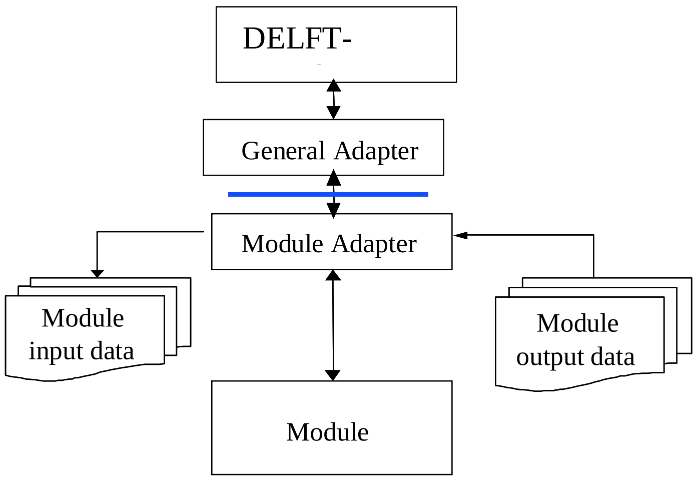


Figure 1 Schematic interaction between the General Adapter and the Module adapter through the published interface.

4.2 Module runtime environment

Most conventions for running modules from DELFT-FEWS through the general adapter concern the runtime environment.

Module input and output

In configuring the general adapter to provide data for use in a specific module, an agreed location (URI path) is given. All files are then written to this location for use by the module adapter. This location, as with all locations configured in the general adapter is a static location, i.e. once configured it does not change from run to run.

In a similar fashion, the module output is expected by the general adapter to be read from an agreed location (URI path). This need not be the same location as that used to make data available to the module.

Running modules from DELFT-FEWS

Running the executable of a third-party module can be defined to follow two principles.

- A single task may be defined, where the general adapter makes a call to a single executable. This is for example a call to the module adapter which then imports data to the module specific format, runs the actual module and subsequently exports data from the module specific format.
- Alternatively, the general adapter can be configured to run three separate tasks in series. The first task is a call to the module adapter to import data to the module native format, the second is a call to the module executable itself and the third and final task is to the module adapter to export results from the native module format to the published interface format.

Of these two options, use of the second option is advocated, as it is then only the general adapter that is required to maintain control over an executable (thread) run by a processor, while the module adapter is only required for reading and writing data. In the first option both the general adapter and the module adapter need to control an executable (thread).

For each task (executable) that is initiated from the general adapter, a working location (URI path) can be defined that is set as the work directory for that executable. This location is again static and does not vary per module run.

Tasks initiated by the general adapter may be given a command line argument. This argument is as with the locations static, and can therefore not contain any dynamic settings (e.g. start of run time etc.).

User interaction

Modules called from the DELFT-FEWS system should in principle be defined such that these require no user interaction through menus and screens. This principle holds for all modules that are run from the system as a requirement in making an automatic forecast. These modules are then typically run on forecasting shell servers, i.e. dedicated servers for module execution. An exception to this rule can be made for modules run on the operator client that require manual interaction. The interface to these is from the point of view of the general adapter exactly the same as any other module. However, the module itself may require user interaction to complete the task required. An example is the interactive correlation utility.

Diagnostics

Should the module be run in a special diagnostic mode, then this can be achieved where applicable though command line arguments.

4.3 Data exchange between General Adapter and Modules

The prime means of communication between external modules and DELFT-FEWS through the general adapter is in the form of data exchange. For running modules in a dynamic situation, this data comprises mainly exchange of dynamic data (see also next sections).

Module input data

Exchange of data is primarily through time series. These are formatted as XML files. The data passed to the modules is in principle already validated by DELFT-FEWS and if applicable gaps in data are filled using the interpolation module. The modules will not be required to perform any data validation or interpolation to fill missing values. Depending on their usage, data series can be passed with or without missing values replaced.

For data series used as boundary conditions to modules, these must typically be complete and missing values are not allowed. The interpolation module (part of DELFT-FEWS) will then be configured such that this is guaranteed.

Data series used for data assimilation, the raw data must be passed. This entails that missing values are not replaced through interpolation. These series will, however, be validated using DELFT-FEWS functionality to remove outliers.

Each data point in a data series is supplied with an additional flag. This flag can be used as a quality flag, indicating for example if a gauge reading is in a range that makes its accuracy questionable. This is referred to as a soft limit, as opposed to a hard limit. Data exceeding hard limits is in principle removed by the validation module.

Module results

Results from external modules may be complete series, but may also contain missing values. For equidistant data series (where a time interval is defined), a time value not included in the file is considered as missing. A missing value constant may also be explicitly defined.

Dealing with module states

Module states can be configured to be exchanged between the central database and the module. These module states are handled in proprietary module formats, and the DELFT-

FEWS system will not be able to make changes in module states. This would require extensive intelligence on the part of the DELFT-FEWS system on the syntax and semantics of module states. States are administered by the DELFT-FEWS system passively, with a time stamp associated with each module state. This time stamp indicates the time for which that state is valid. On running a module, the state associated with the start time of that run is made available through the module adapter.

For each module a default state is also administered which can be used in for example cold start conditions. This default state does not have a fixed time stamp. Besides this defaults state, the system can also administer a number of scenario states in the same way, where these do not have a definite time stamp. For some modules, there is a requirement that when the state is used to initiate a run, then that state must have within it the exact time stamp for which it is valid. If this is the case then the module adapter is responsible for setting that time stamp.

On completion of or during a module run module states can be exchanged through the general adapter. Each state carries in the XML file definition the time stamp with which it is associated.

The DELFT-FEWS system itself does not allow for amending values in system states. If a value from an external source (e.g. SMD values, reservoirs levels) is required to be amendable in the module state, then this is passed to the module a (non-equidistant) time series. The *module adapter* is responsible for incorporating these values into the module state in a meaningful way.

Module diagnostics

Communication of module diagnostics is through the applicable published interface formats. The general adapter starts a task, and on completion reads the output of that task. There is no provision for intermediate communication.

5 General Remarks

The following remarks apply to all parts of this document:

- All date and time information is in GMT unless stated otherwise. Note that the ISO 8601 specification allows for specifying the timezone within the time type itself the use thereof is depreciated. Please use the timeZone element instead
- If no recognised geographical datum is given all location data is assumed to be in a local (metric) grid
- The consistency of links (i.e. between branches and lateral inputs) should be checked before importing the data. DELFT-FEWS imports the data and links as is.
-

6 Introduction to File Format Specifications

In this part the layout of all the different file types (used by the general adapter) is briefly described. The *full description* is given in the Appendix which holds the schema documentation. If the schema documentation is not complete (or clear), the actual schema (XSD) files should be consulted. The supported ASCII grids and binary grids are not described in the schema files. The format descriptions of these files are contained within this document in the Appendices.

Please note that only a subset of the presented files formats may be needed for a particular application. It's prime communication with external forecasting modules is through dynamic data such as time series, model states and diagnostics (see below). The published interface formats have been defined to cover a wider range of data types, but these need not be used in all cases – i.e. there is no requirement to allow communication of all data types. Indeed there is currently no module that caters for the full range of formats.

In exchanging data between a module and NFFS, a prioritisation of data types can be identified.

Priority	**Data Type**	**Comment**
1	Time Series Module states Diagnostics	sufficient to cover most modules used in flood forecasting systems, including e.g. Mike-11 and NAM
1 (a)	Time series of grid data	additionaly required for modules with 2D I/O formats (e.g inundation codes)
2	Parameters	Module parameters may be passed where the module allows calibration through the NFFS calibration facilities
2 (a)	Longitudinal profile data	For hydrodynamic modules longitudinal data types may be passed – not a strict requirement.
3	Static data (cross sections, branches, etc)	This data is not required for operational forecasting. In exceptional cases data such as on branches may be required (for display purposes), but this need not be passed through the adapter and can be configured as appropriate.

7 Dynamic data

7.1 Time Series

Time series data represent data collected over a given period of time at a specific location. DELFT-FEWS can read and write time series in XML format (see fileformats.xsd) described below. Time series files can contain both equidistant times series and non-equidistant series. Multiple time series can be stored in a single file. All time and date information is given in GMT unless otherwise stated. The default (and preferred) missing value definition is NaN. Quality flags are constructed on a philosophy of two qualifiers. The first described the origin of the data and the second the quality.

Possible origins of data are:

1. Original: This entails the data value is the original value. It has not been amended by DELFT-FEWS
2. Completed: This entails the original value was missing and was replaced by a non-missing value.
3. Corrected: This entails the original value was replaced with another non-missing value.

Possible qualifiers are:

1. Reliable: Data is reliable and valid
2. Doubtful: The validity of the data value is uncertain
3. Unreliable: The data value is unreliable and cannot be used.

See Appendix 10D for a tabular presentation of the quality flags.

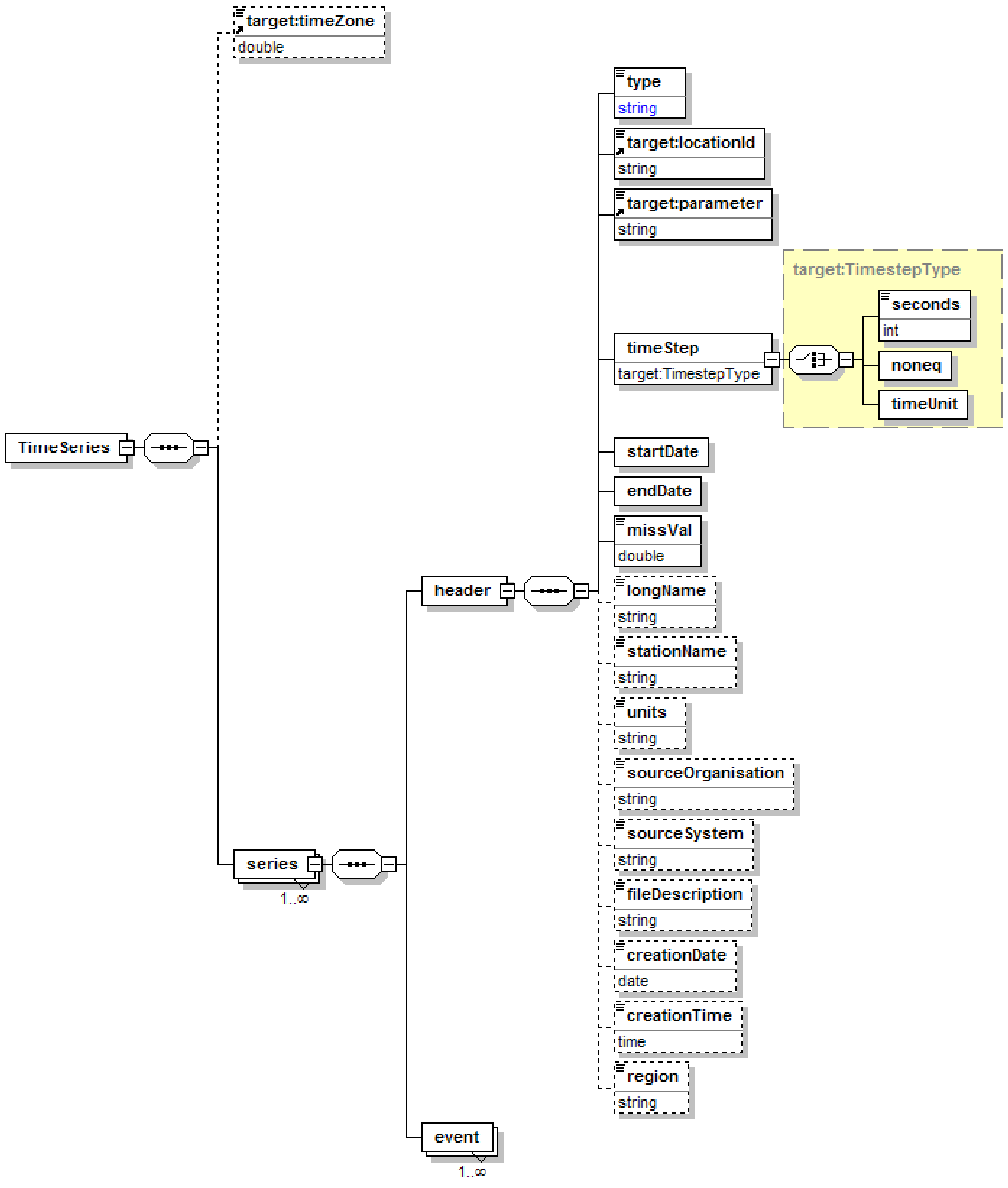
schema file, root element	**Example file**
`pi_timeseries.xsd, Timeseries`	`Timeser.xml`

The XML file holds a header containing metadata for the time series. Time series may either be equidistant or non-equidistant. This is indicated in the timestep element.

Example:
```
<?xml version="1.0" encoding="UTF-8"?>
<TimeSeries version="1.1" xmlns="http://www.wldelft.nl/fews" xmlns:target="http://www.wldelft.nl/fews"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.wldelft.nl/fews
pi_timeseries.xsd">
    <series>
        <header>
            <type>accumulative</type>
            <locationId>Rhine_99_1</locationId>
            <parameter>Precipitation</parameter>
            <timeStep>
                <seconds>3600</seconds>
            </timeStep>
            <startDate date="1967-08-13" time="14:00:00"/>
            <endDate date="1967-08-13" time="15:00:00"/>
            <missVal>-999</missVal>
            <longName>Rockenau</longName>
```

```
                <stationName>Rockenau</stationName>
                <units>mm</units>
                <sourceOrganisation>BfG</sourceOrganisation>
                <sourceSystem>DWD-system3</sourceSystem>
                <fileDescription>Very short series for Rockenau</fileDescription>
                <creationDate>2003-01-11</creationDate>
                <creationTime>14:00:00</creationTime>
            </header>
            <event date="1967-08-13" time="14:00:00" value="3.1" flag="0"/>
            <event date="1967-08-13" time="15:00:00" value="-999"/>
        </series>
        <series>
            <header>
                <type>instantaneous</type>
                <locationId>Rhine_99_3</locationId>
                <parameter>Discharges</parameter>
                <timeStep>
                    <noneq/>
                </timeStep>
                <startDate date="1967-08-13" time="14:00:00"/>
                <endDate date="1967-08-13" time="15:00:00"/>
                <missVal/>
                <longName>Rockenau</longName>
                <stationName>Rockenau</stationName>
                <units>m3/s</units>
                <sourceOrganisation>BfG</sourceOrganisation>
                <sourceSystem>DWD-system3</sourceSystem>
                <fileDescription>Very short series </fileDescription>
                <creationDate>2003-01-11</creationDate>
                <creationTime>14:00:00</creationTime>
            </header>
            <event date="1967-08-13" time="14:00:00" value="3.1" flag="0"/>
            <event date="1967-08-13" time="15:00:00" value="NaN"/>
            <event date="1967-08-13" time="18:00:00" value="7.1"/>
        </series>
    </TimeSeries>
```

The structure of the XML schema is given in the figures below. Note that attributes are not displayed in the figures. Consult the full schema (+documentation) for details.


target:timeZone
double
TimeSeries
series
1..∞
header
event
1..∞
type
string
target:locationId
string
target:parameter
string
timeStep
target:TimestepType
target:TimestepType
seconds
int
noneq
timeUnit
startDate
endDate
missVal
double
longName
string
stationName
string
units
string
sourceOrganisation
string
sourceSystem
string
fileDescription
string
creationDate
date
creationTime
time
region
string

7.2 (Time series) Location

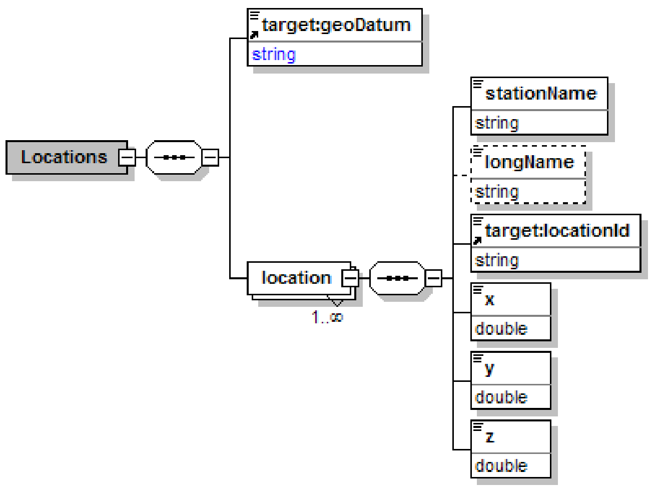
Time series location data files identify points at which specific times series exist. A point is represented by X, Y and Z co-ordinates. If no geographical datum is supplied *LOCAL* is assumed.

schema file, root element	example file
`pi_locations.xsd, Locations`	`location.xml`

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<Locations xmlns="http://www.wldelft.nl/fews" xmlns:target="http://www.wldelft.nl/fews"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.wldelft.nl/fews
pi_locations.xsd" version="1.1">
    <geoDatum>WGS-1984</geoDatum>
    <location>
        <stationName>Mozel 34</stationName>
        <locationId>LocationA</locationId>
        <x>3.2</x>
        <y>3.1</y>
        <z>100</z>
    </location>
    <location>
        <stationName>Hupselse beek</stationName>
        <locationId>LocationB</locationId>
        <x>5.1</x>
        <y>3.9</y>
        <z>100</z>
    </location>
    <location>
        <stationName>No</stationName>
        <locationId>LocationC</locationId>
        <x>4.1</x>
        <y>3.1</y>
        <z>100</z>
    </location>
</Locations>
```

The structure of the XML schema is given in the figures below. Note that attributes are not displayed in the figures. Consult the full schema (+documentation) for details.



7.3 Map (stacks) data (grids)

Maps stacks are time series of grid data. Map stacks that hold one time step are treated as static data. The maps themselves are stored in either binary or ASCII file(s) but the description and the information needed to import them into the DELFT-FEWS database is stored in an XML file.

Each time step is stored in a separate ASCII/PCRaster based file or in a single USGS BIL file. In the former the order of the maps is determined by the file extension and up to the last four characters of the first part of the filename:

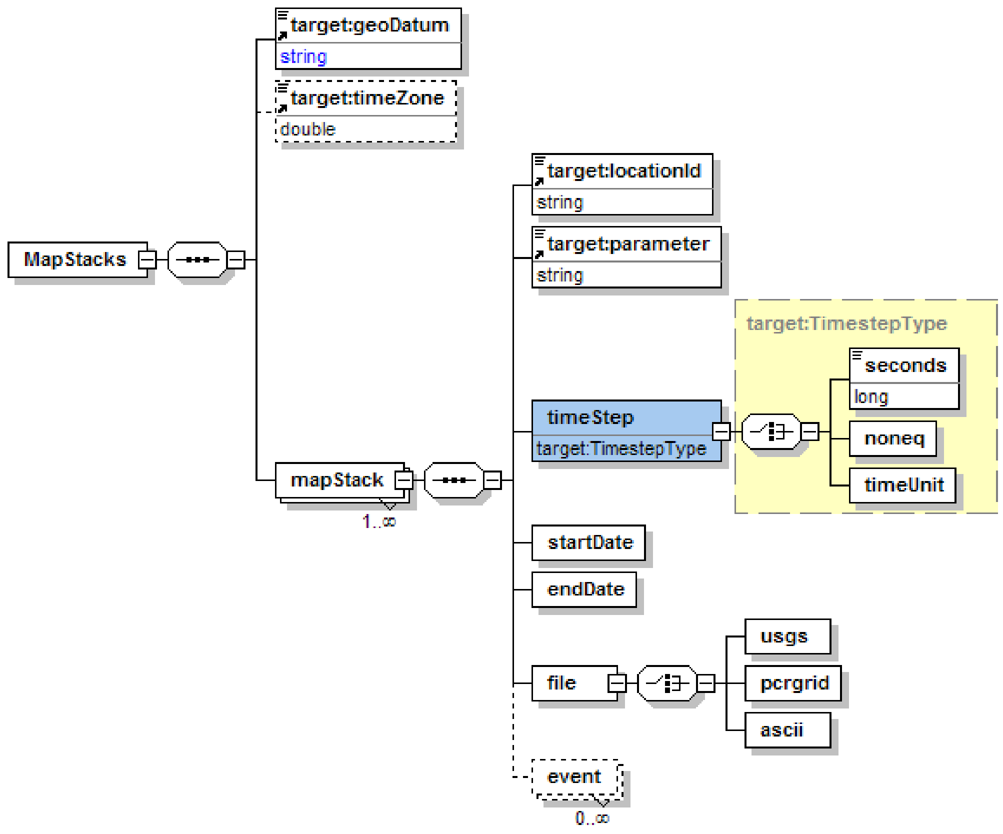
(eg. `evap0000.001`, `evap0000.002` …. `evap0000.999`, `evap0001.000`). This limits the size of a stack of maps to 9999999.

schema file, root element	**example file**
`pi_mapstacks.xsd, Mapstacks`	`mapstack.xml`

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<MapStacks xmlns="http://www.wldelft.nl/fews" xmlns:target="http://www.wldelft.nl/fews/interface-0.9"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.wldelft.nl/fews
pi_mapstacks.xsd" version="1.1">
    <geoDatum/>
    <mapStack>
        <locationId>test1</locationId>
        <parameter>vpd</parameter>
        <timeStep>
            <noneq/>
        </timeStep>
        <startDate date="2003-09-09" time="18:00"/>
        <endDate date="2003-09-09" time="20:00"/>
        <file>
            <usgs file="file://allvpd.bil" header="file://allvpd.hdr"/>
        </file>
        <!--Three maps are defined, starting at 18:00, ending at 20:00-->
        <event date="2003-09-09" time="18:00"/>
        <event date="2003-09-09" time="19:00"/>
        <event date="2003-09-09" time="20:00"/>
    </mapStack>
</MapStacks>
```

The structure of the XML schema is given in the figures below. Note that attributes are not displayed in the figures. Consult the full schema (+documentation) for details.



The following three formats are allowed for the actual grids:

1. ESRI .ASC ASCII file grids (see appendix B3)
2. USGS BIL, BIP, BSQ files (with a separate header, see appendix B2)
3. PCRaster maps (see appendix B1)

7.4 Module state data

Modules can have their state (i.e. all the data and parameters the module needs to restart) saved by DELFT-FEWS. Many different solutions are in use by current models. As such, DELFT-FEWS regards model state data as a black box, it is not parsed or interpreted by the system. Programs must specify a list of files and/or directories that DELFT-FEWS can save (and later restore).

The model data state file is an XML file in which a read and write location are specified for each file/directory. If the read location is a directory, the write location needs to be a directory also (and v.v).

Each state must include a timestamp indicating the date and time for which that state is valid. A module ID is included to identify to which module the state pertains.

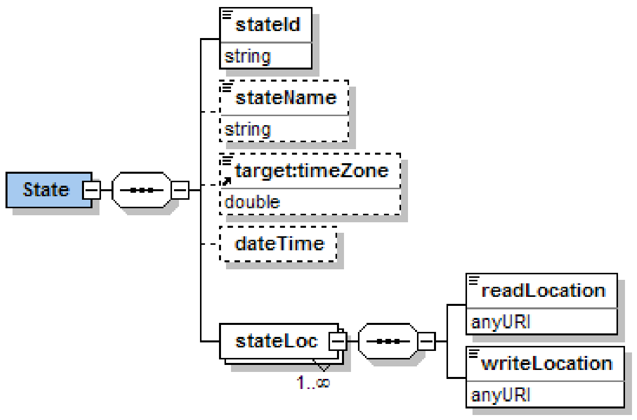
The readLocation is the location the *module adapter reads from*, the writeLocation is the location the *module adapter writes to*. Consequently the general adapter will read the data from the writeLocation and write the data to the readLocation.

Schema file, root element	**example file**
`pi_state.xsd, State`	`state.xml`

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<State xmlns="http://www.wldelft.nl/fews" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.wldelft.nl/fews pi_state.xsd" version="1.1">
    <stateId>sum23</stateId>
    <stateName>Summer 1995</stateName>
    <dateTime date="2003-06-06" time="10:00"/>
    <stateLoc type="file">
        <readLocation>file://d:/exp/sbek.mda</readLocation>
        <writeLocation>file://d:/imp/sbek.mda</writeLocation>
    </stateLoc>
    <stateLoc type="directory">
        <readLocation>file://d:/HBV</readLocation>
        <writeLocation>file://d:/HBV</writeLocation>
    </stateLoc>
</State>
```

The structure of the XML schema is given in the figures below. Note that attributes are not displayed in the figures. Consult the full schema (+documentation) for details.


State
stateId
string
stateName
string
target:timeZone
double
dateTime
stateLoc
1..∞
readLocation
anyURI
writeLocation
anyURI

7.5 Longitudinal profile

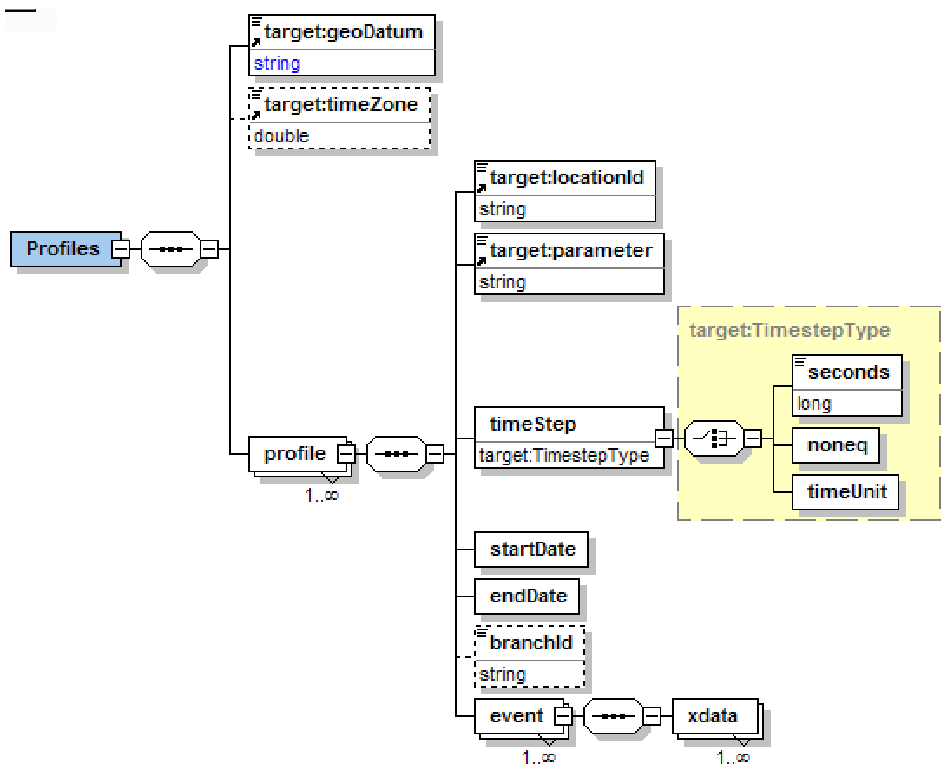
A Longitudinal profile describes a profile of (part of) a branch. For each point in a profile a chainage and a value are defined. Multiple profiles can be specified in a single file. Each profile can hold several timesteps, i.e. several instances of the same profile at different times.

schema file, root element	**Example file**
`pi_profiles.xsd,Profiles`	`Prof.xml`

Example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Profiles xmlns="http://www.wldelft.nl/fews" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.wldelft.nl/fews pi_profiles.xsd" version="1.1">
    <geoDatum/>
    <profile>
        <locationId>some id</locationId>
        <parameter>water level</parameter>
        <timeStep>
            <seconds>3600</seconds>
        </timeStep>
        <startDate date="2003-09-09" time="13:00"/>
        <endDate date="2003-09-09" time="14:00"/>
        <event date="2003-09-09" time="13:00">
            <xdata x="12" y="13" z="0" chainage="356" value="24.9"/>
        </event>
        <event date="2003-09-09" time="14:00">
            <xdata x="12" y="13" z="0" chainage="356" value="24.9"/>
        </event>
    </profile>
</Profiles>
```

The structure of the XML schema is given in the figures below. Note that attributes are not displayed in the figures. Consult the full schema (+documentation) for details.


target:geoDatum
string
target:timeZone
double
Profiles
target:locationId
string
target:parameter
string
target:TimestepType
seconds
long
noneq
timeUnit
timeStep
target:TimestepType
profile
1..∞
startDate
endDate
branchId
string
event
1..∞
xdata
1..∞

7.6 Module parameters

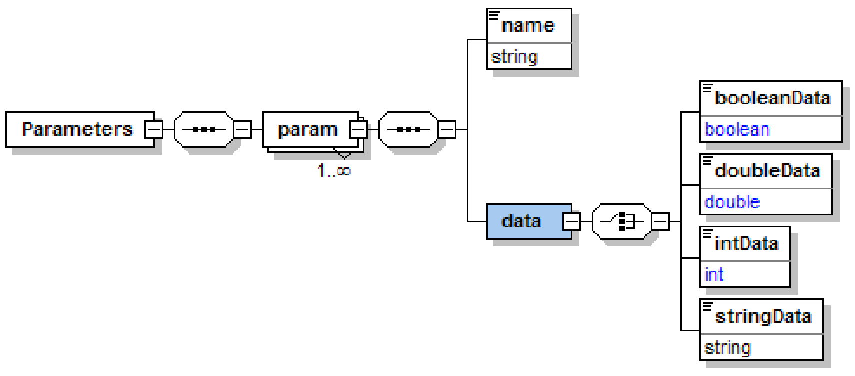
DELFT-FEWS can read and write module parameters in XML format. The format is described in the schema. Each file can contain a number of parameters that are (optionally) combined in parameter sets. A parameter can be a floating point value (with a specified minimum and maximum value), an integer value (also with a specified minimum and maximum value) or a Boolean value (either 0 or 1, false or true). Double long and Boolean values can have an allow adjust parameter that indicates if DELFT-FEWS may adjust the parameter in some sort of fitting exercise. String parameters can be used to store specific module information.

schema file, root element	**example file**
`pi_parameters.xsd, parameters`	`params.xml`

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<Parameters xmlns="http://www.wldelft.nl/fews" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.wldelft.nl/fews pi_prameters.xsd" version="1.1">
    <param>
        <name>Van genuchten Alpha</name>
        <data>
            <doubleData allowAdjust="1" maxVal="0.6" minVal="0.01">0.12</doubleData>
        </data>
    </param>
    <param>
        <name>Van genuchten n</name>
        <data>
            <doubleData allowAdjust="1" maxVal="1.9" minVal="0.6">1.2</doubleData>
        </data>
    </param>
    <param>
        <name>Nr itterations</name>
        <data>
            <intData allowAdjust="0" maxVal="500" minVal="3">300</intData>
        </data>
    </param>
</Parameters>
```

The structure of the XML schema is given in the figures below. Note that attributes are not displayed in the figures. Consult the full schema (+documentation) for details.



7.7 Lookup tables

To relate parameters in lookup tables two or three parameter lookup tables can be used (e.g. stage height to discharge tables). Each XML file can contain one or more relations. Optional information such as extrapolation type and the validity (time) of the relation can be added.

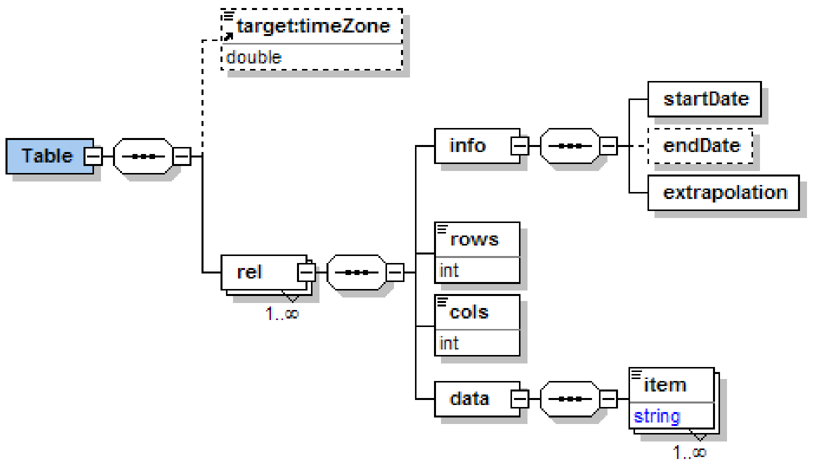
schema file, root element	**example file**
`pi_table.xsd, table`	`lookup.xml`

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<Table xmlns="http://www.wldelft.nl/fews" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.wldelft.nl/fews
pi_table.xsd" version="1.1">
    <rel>
        <info>
            <startDate date="2003-09-09" time="13:00"/>
            <extrapolation type="non"/>
        </info>
        <!-- The following defines a 2 rows by three columns matrix -->
        <rows>2</rows>
        <cols>3</cols>
        <data>
            <!--The items are processed per row-->
            <item type="double">12</item>
            <item type="double">30</item>
            <item type="string">Waterlooo station</item>
            <item type="double">13</item>
            <item type="double">40</item>
            <item type="string">Westeinder</item>
        </data>
        <!-- The end-result, in tabular form should look like this:

            col1      col2      col3
            ==========================
            12        30       Waterloo station
            13        40       Westeinder
        -->
    </rel>
</Table>
```

The structure of the XML schema is given in the figures below. Note that attributes are not displayed in the figures. Consult the full schema (+documentation) for details.


target:timeZone
double
Table
rel
1..∞
info
startDate
endDate
extrapolation
rows
int
cols
int
data
item
string
1..∞

7.8 Module diagnostics output

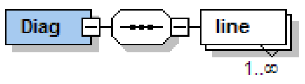
DELFT-FEWS uses the module diagnostics output format to report the results from modules to the user. The general adapter can also use these files. The diagnostics files can hold several warning levels attached to a line (message):
3 = info (information, all is well, e.g. :"SOBEK: program ended") 2 = warn (warning information. e.g. "SOBEK: high number of iterations") 1 = critical (critical problems. e.g. "SOBEK: no convergence") 0 = crash (full module crash. e.g. "SOBEK: ooops, what now?").

schema file, root element	**example file**
`pi_diag.xsd, Diag`	`diag.xml`

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<Diag xmlns="http://www.wldelft.nl/fews" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.wldelft.nl/fews pi_diag.xsd" version="1.1">
    <line level="3" description="HVB Starting at 12:00 10 Nov 2002"/>
    <line level="1" description="No improvement in optimisation"/>
    <line level="3" description="Run ended at 12:03 10 Nov 2002"/>
    <!--Each line/mesasage contains the actual text and a warning level.
each message/line has a level attached to it
3 = info (information, all is well, e,g, :"SOBEK: program ended")
2 = warn (warning information.
e.g. "SOBEK: high number of iterations")
1 = critical (critical problems. e.g. "SOBEK: no convergence")
0 = crash (full module crash. e.g. "SOBEK: ooops, what now?")
All levels higher than 3 are regarded as non-essential (debug) information-->
</Diag>
```

The structure of the XML schema is given in the figures below. Note that attributes are not displayed in the figures. Consult the full schema (+documentation) for details.



7.9 Polygons

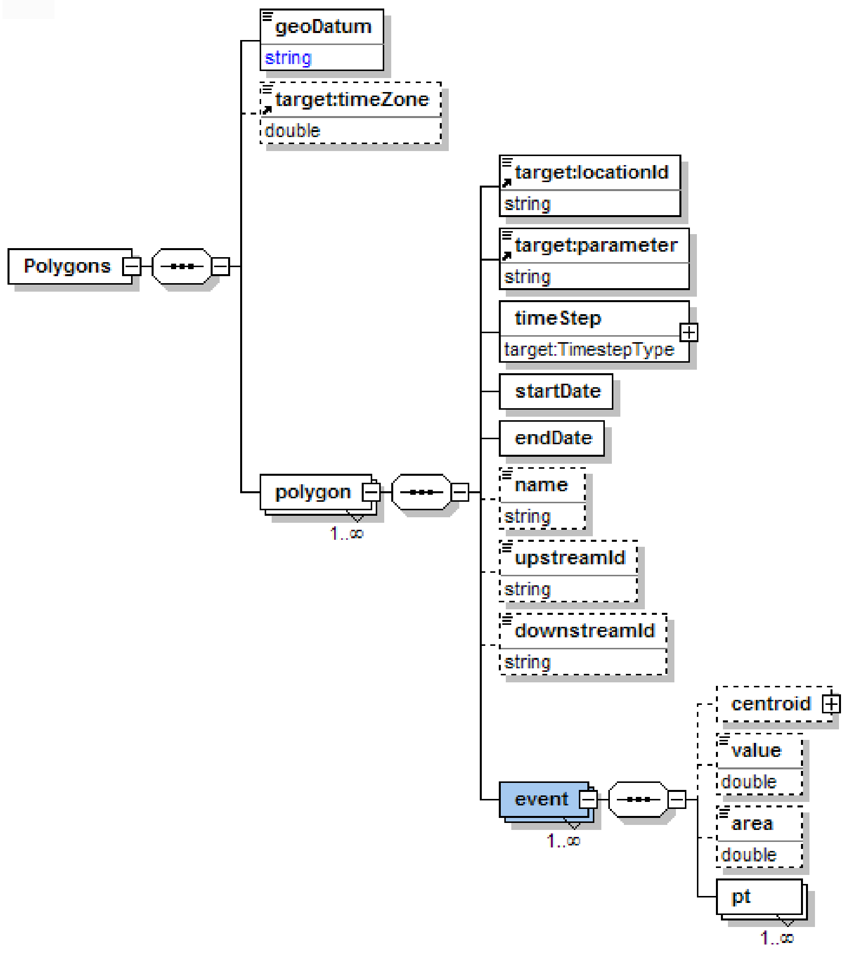
Polygon data identifies a specific area. For example, a catchment that contributes water to a specific branch. This data type consists of several elements, including a list of X and Y point locations that form the vertices of a polygon feature. The first and last points in this list are joined to form a closed polygon. Optionally a polygon centroid can be given. Usually this is the point to which precipitation input to (lumped) models must be interpolated.
Each polygon element can hold several events. As such, polygons can also be dynamic elements.

schema file, root element	example file
`pi_polygons.xsd, Polygons`	`polygons.xml`

Example:

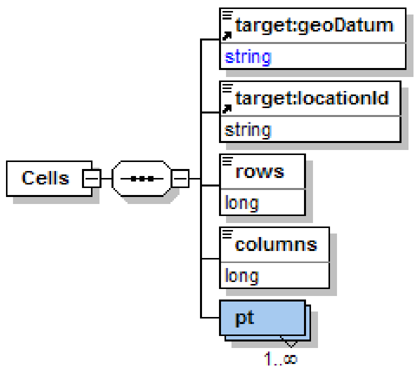
```
<?xml version="1.0" encoding="UTF-8"?>
<Polygons xmlns="http://www.wldelft.nl/fews" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.wldelft.nl/fews
pi_polygons.xsd" version="1.1">
    <geoDatum>LOCAL</geoDatum>
    <timeZone>0.0</timeZone>
    <polygon>
        <locationId>String</locationId>
        <parameter>String</parameter>
        <timeStep>
            <seconds>3600</seconds>
        </timeStep>
        <startDate date="1967-08-13" time="13:00:00"/>
        <endDate date="1967-08-13" time="14:00:00"/>
        <name>Flood Rhine</name>
        <upstreamId>rh2</upstreamId>
        <event date="1967-08-13" time="13:00:00">
            <pt x="3.14159265358979" y="3.14159265358979" z="3.14159265358979" mark="1"/>
            <pt x="3.14159265358979" y="3.14159265358979" z="3.14159265358979" mark="2"/>
        </event>
        <event date="1967-08-13" time="14:00:00">
            <pt x="3.14159265358979" y="3.14159265358979" z="3.14159265358979" mark="0"/>
            <pt x="3.14159265358979" y="3.14159265358979" z="3.14159265358979" mark="0"/>
        </event>
    </polygon>
</Polygons>
```

The structure of the XML schema is given in the figures below. Note that attributes are not displayed in the figures. Consult the full schema (+documentation) for details.


geoDatum
string
target:timeZone
double
target:locationId
string
target:parameter
string
timeStep
target:TimestepType
startDate
endDate
Polygons
polygon
1..∞
name
string
upstreamId
string
downstreamId
string
centroid
value
double
event
1..∞
area
double
pt
1..∞

8 Static data

8.1 Grid cell center points



8.2 Branch data

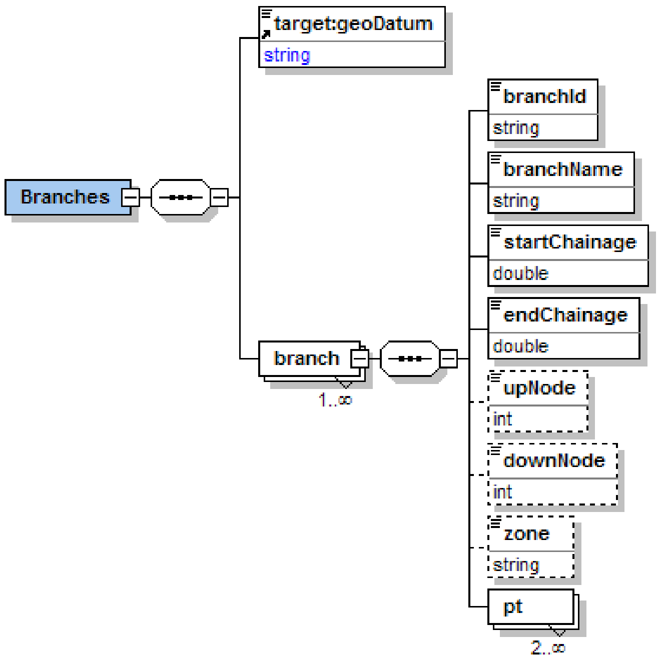
Branch data is used to represent linear stretches of rivers, streams and canals. Each branch has an unique identifier (choosen by the module) a name, a start and end-chainage field, a link to an upper node and a link to a downstream node. A series of X, Y points defines the actual branch, the first point being the upstream node, the last point the downstream node. This date type is based on the EUROTAS ICM definition.

schema file, root element	example file
`pi_branches.xsd, Branches`	`branches.xml`

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<Branches xmlns="http://www.wldelft.nl/fews" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.wldelft.nl/fews
pi_branches.xsd" version="1.1">
    <geoDatum>LOCAL</geoDatum>
    <branch>
        <branchId>rhone3_24</branchId>
        <branchName>A branch</branchName>
        <startChainage>0</startChainage>
        <endChainage>13</endChainage>
        <pt x="12" y="31" chainage="0"/>
        <pt x="18" y="36" chainage="9"/>
        <pt x="22" y="38" chainage="12"/>
        <pt x="24" y="33" chainage="18"/>
        <pt x="37" y="0" chainage="13"/>
    </branch>
    <branch>
        <branchId>rhone3_25</branchId>
        <branchName>Another branch</branchName>
        <startChainage>1</startChainage>
        <endChainage>3</endChainage>
        <pt x="37" y="0" chainage="0"/>
        <pt x="40" y="40" chainage="1"/>
        <pt x="45" y="55" chainage="2"/>
    </branch>
</Branches>
```

The structure of the XML schema is given in the figures below. Note that attributes are not displayed in the figures. Consult the full schema (+documentation) for details.


target:geoDatum
string
Branches
branch
1..∞
branchId
string
branchName
string
startChainage
double
endChainage
double
upNode
int
downNode
int
zone
string
pt
2..∞

8.3 Cross-sections

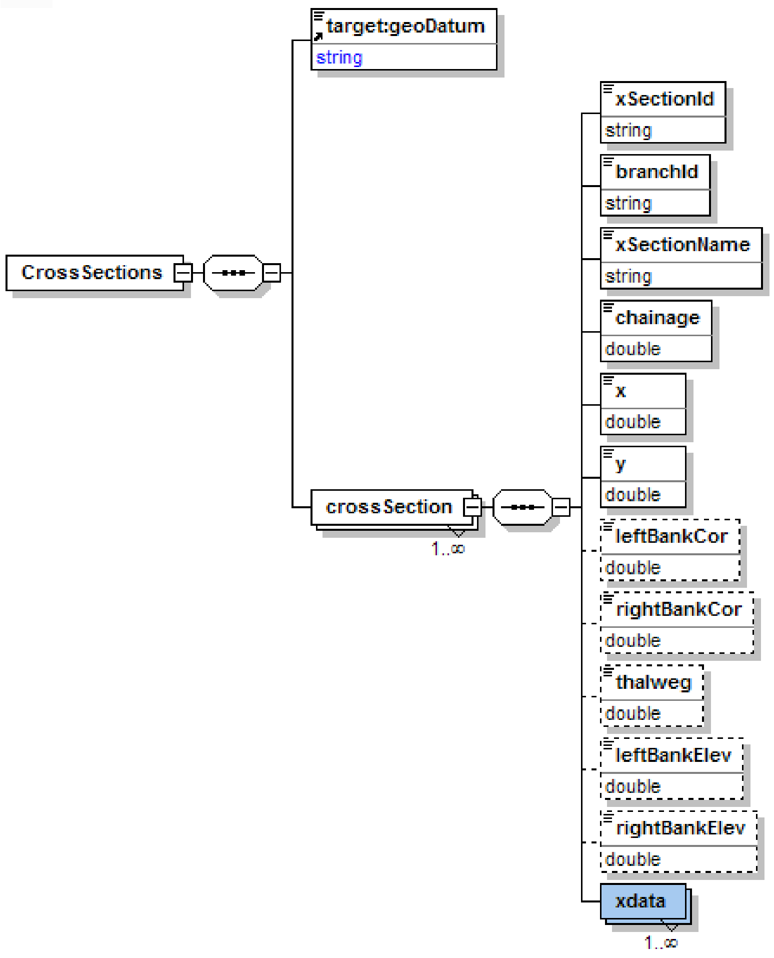
Cross sections are measured locations along a branch. As such they are linked to a specific branch (the branchId). Cross sections are always considered as being so-called YZ cross sections. This entails that the geometrical form of the cross section is described through a set of cross section co-ordinates, where the first co-ordinate represents the distance across the cross section, where this co-ordinate increases from left to right (i.e. lowest co-ordinate is on the left bank) and the second co-ordinate represents the elevation with respect to a common reference level.

schema file, root element	**example file**
`pi_crosssections.xsd, CrossSections`	`xsection.xml`

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<CrossSections xmlns="http://www.wldelft.nl/fews" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://www.wldelft.nl/fews pi_crosssections.xsd" version="1.1">
    <geoDatum>WGS-1984</geoDatum>
    <crossSection>
        <xSectionId>PR2</xSectionId>
        <branchId>Mameyes2</branchId>
        <xSectionName>Puento Rotto</xSectionName>
        <chainage>10</chainage>
        <x>18.3</x>
        <y>1.0</y>
        <xdata csy="21" z="18" roughness="0.4" mark="1"/>
        <xdata csy="22" z="17" roughness="0.4" mark="2"/>
        <xdata csy="245" z="18" roughness="0.4" mark="3"/>
    </crossSection>
</CrossSections>
```

The structure of the XML schema is given in the figures below. Note that attributes are not displayed in the figures. Consult the full schema (+documentation) for details.


target:geoDatum
string
CrossSections
crossSection
1..∞
xSectionId
string
branchId
string
xSectionName
string
chainage
double
x
double
y
double
leftBankCor
double
rightBankCor
double
thalweg
double
leftBankElev
double
rightBankElev
double
xdata
1..∞

9 Lateral inputs

Lateral inputs can be added to the river network topology. These can be used to model tributaries not explicitly modelled as a branch. The discharge allocated to the lateral input is typically a time series, or a given constant value.

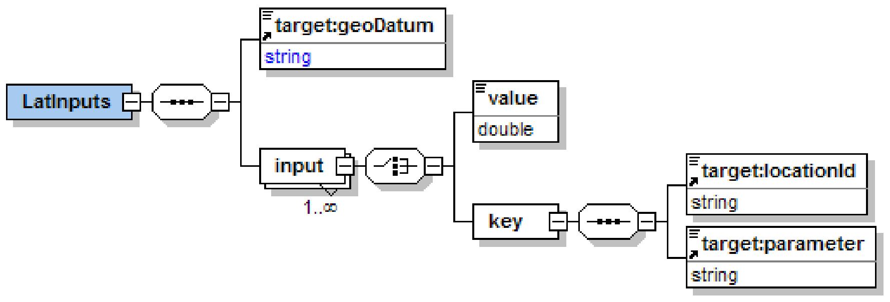
Definition of lateral input is done a single XML file, giving the location of the lateral on the river network in the same way as was the case for the cross sections.

schema file, root element	example file
`pi_latinputs.xsd, LatInputs`	`latinput.xml`

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<LatInputs xmlns="http://www.wldelft.nl/fews" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.wldelft.nl/fews
pi_latinputs.xsd" version="1.1">
    <geoDatum>LOCAL</geoDatum>
    <input latId="Dinkel1" x="3.14" y="9.19" branchId="Regge56" chainage="200">
        <value>12</value>
    </input>
    <input latId="Dinkel2" x="3.14" y="9.19" branchId="Regge56" chainage="300">
        <value>40</value>
    </input>
</LatInputs>
```

The structure of the XML schema is given in the figures below. Note that attributes are not displayed in the figures. Consult the full schema (+documentation) for details.



10 List of files

The fileformats.xsd is a container schema that includes the schemas below.

Basename	**Extension**	**Root element**	**Page**	**Comment**
`pi_parameters`	`xsd`	parameters	**Error! Bookmark not defined.**	Module parameters
`pi_timeseries`	`xsd`	timeseries	**Error! Bookmark not defined.**	Time series
`pi_latinputs`	`xsd`	latinputs	**Error! Bookmark not defined.**	Lateral inputs
`pi_locations`	`xsd`	locations	**Error! Bookmark not defined.**	Time series location
`pi_mapstacks`	`xsd`	mapstacks	**Error! Bookmark not defined.**	Map stacks
`pi_state`	`xsd`	state	**Error! Bookmark not defined.**	Module state
`pi_table`	`xsd`	table	**Error! Bookmark not defined.**	Lookup tables
`pi_diag`	`xsd`	diag	**Error! Bookmark not defined.**	Module diagnostics
`pi_branches`	`xsd`	branches	**Error! Bookmark not defined.**	Branches
`pi_crosssections`	`xsd`	crosssections	**Error! Bookmark not defined.**	Cross sections
`pi_polygons`	`xsd`	polygons	**Error! Bookmark not defined.**	polygon boundary
`pi_profiles`	`xsd`	profiles	**Error! Bookmark not defined.**	Longitudinal profile

Basename	**Extension**	**Root element**	**Page**	**Comment**
`pi_cells`	`xsd`	cells	**Error! Bookmark not defined.**	grid cell centre point data

Appendices

A XML Schema listings

The table below lists the various XML Schemas. Page number refers to the page in Appendix E where the schema is defined.

Schema	**page in Appendix E**
Contents	1
Schema fileformats.xsd	2
element Branches	3
element Cells	6
element CrossSections	7
element Diag	10
element geoDatum	11
element LatInputs	12
element locationId	13
element Locations	14
element MapStacks	16
element parameter	19
element Parameters	20
element Polygons	22
element Profiles	27
element State	29
element Table	31
element TimeSeries	34
element timeZone	39
complexType TimestepType	40

B Grid data file formats

B.1 PCRaster

PCRaster is a unique raster based GIS package ideally suited for dynamic modelling. Information on the PCRaster native format (and a free version, including tools for conversion) can be obtained from http://www.pcraster.nl.

B.2 USGS BIL / BSQ / BIP

These files consist of an ASCII header file (with content information) and a binary data file. The ASCII description files support the following formats:

- Band sequential (BSQ) multiband images
- Band interleaved by line (BIL) multiband images
- Band interleaved by pixel (BIP) multiband images

In order to support time steps an addition has been made to the .hdr file. A USGS BIL, BIP and BSQ file always have a *.hdr header file that contains the settings for the dataset. So as to add support for time data in these files the following changes must be made:

- Add a line to the header file with the number of time blocks: for example nBlocks 3.

- The nBands property in the header file represents the number of available parameters.

Creating a dynamic bil file:

*Create the binary file in BSQ, BIL or BIP format. If you use for example ArcView to create *.BSQ binary files for each time step, then you can use the DOS copy command to add files together as follows:*

Copy /B timestep1.bsq /B + timestep2.bsq /B + timestep3.bsq /B + … timestepN.bsq /B destination.bsq /B
For more information on the copy command, go to the Start Run menu and type copy /? and press enter.

Example of a header file:

```
ByteOrder I
Layout BIL
nRows 2
nCols 2
nBands 1
nBlocks 1
nBits 32
BandRowBytes 8
```

```
TotalRowBytes 8
BandGapBytes 0
NoData -999
ULXmap 163900
ULYmap 522900
Xdim 10000
Ydim 10000
```

B.3 Image file

The binary image file for the BIL/BIP/BSQ image format is merely a bit stream of the image data. How the image data is arranged in that bit stream defines whether it is a BIL, BIP, or BSQ image.

Band interleaved by line data stores pixel information band by band for each line, or row, of the image. For example, given a three-band image, all three bands of data are written for row one, all three bands of data are written for row two, and so on, until the total number of rows in the image is reached.

Band interleaved by pixel data is similar to BIL data, except that the data for each pixel is written band by band. For example, with the same three-band image, the data for bands one, two, and three is written for the first pixel in column one; the data for bands one, two, and three is written for the first pixel in column two; and so on. Band sequential format stores information for the image one band at a time. In other words, data for all pixels for band one is stored first, then data for all pixels for band two, and so on.

For further information see:
`http://www.esri.com/library/whitepapers/pdfs/eximgav.pdf`

B.4 ESRI ASCII

ASCII grids are stored in a format compatible with ESRI (and many other) software. The ASCII raster file format is a simple format that can be used to transfer raster data between various applications. The header data includes the following keywords and values:

- ncols - number of columns in the data set.
- nrows - number of rows in the data set.
- xllcenter or xllcorner - x-coordinate of the centre or lower-left corner of the lower-left cell.
- yllcenter or yllcorner - y-coordinate of the centre or lower-left corner of the lower-left cell.
- cellsize - cell size for the data set.
- nodata_value - value in the file assigned to cells whose value is unknown. This keyword and value is optional. The nodata_value defaults to -9999.

The first row of data is at the top of the data set, moving from left to right. Cell values should be *delimited by spaces*. No carriage returns/linefeeds are necessary at the end of each row in the data set. The number of columns in the header is used to determine when a new row begins. The number of cell values must be equal to the number of rows times the number of columns.

Example:

```
ncols 4
nrows 3
xllcorner 175208.9306
yllcorner 320440.9027
cellsize 25
NODATA_value -9999
12    13    14    15
12    13    14    15
12    14    14    15
```

C EA Schema to PI mapping

The EA Time Series Schema established in draft from for hydrometric data can be mapped to the time series XML schema established as a part of the published interface. This mapping is illustrated in the table below (* indicate mandatory fields).

NFFS Published Interface	**EA Hydrometric Proposed Format**	**Comment**
Header	**HydrometricData**	
sourceOrganisation	sourceOrganisation	
sourceSystem	sourceSystem	
fileDescriprion	fileDescriprion	
creationDate	creationDate	
creationTime	creationTime	
	Station	
region	region	Optional. May be useful if coding used is not unique across regions
stationName	stationName	
longname		Optional long descriptive name of station
locationid	stationReference	Compulsory
geodatum		Identifies geographic datum
location	ngr	Uses "geodatum" field to allow different datums - ngr is OS1936
	SetofReadings	
parameter	parameter *	compulsory
type*	dataType *	Enumeration supports only Accumulative of Instantaneous - interval specified later
units*	units *	Optional string identifying units - ISO standards should be adhered to
startdate	startDate	field for date
startdate	startTime	field for time
endate	endDate	field for date
endate	endTime	field for time
missval	invalidNumber	
	dayOrigin	not included in published interface
	readingsPerDay	not included in published interface. Information contained in other fields
time step*		Time step in seconds (used for equidistant series)
Event	**Reading**	
data*	date *	field for date
time*	time *	field for time
value*	Reading*	field for value of reading
flag	quality	Only single flag considered - could be used for quality flag
	quality2	not included in Published Interface
	highLow	not included in Published Interface, may be combined with flag
	Comment	not included in Published Interface
	startDate	not included in Published Interface

NFFS Published Interface	**EA Hydrometric Proposed Format**	**Comment**

	startTime
	endDate
	endTime
		not included in Published Interface
		not included in Published Interface
		not included in Published Interface

D Quality flags

Following the discussion between Delft Hydraulics and CEH & Wallingford Software on Friday 2 May 2003 in Wallingford on providing an enumeration of quality flags as a part of the SIS, a proposed quality flag enumeration is given in the table below. As described in the timeseries XML schemas provided we have made allowance for a single quality flag. This is contrary to the EA XML interchange formats which provide for three separate quality flags. A single flag seems more appropriate to the level of communication we are dealing with and should avoid ambiguity. The flags are single byte values and are incorporated in the XML Schema for time series. Quality flags are provided for each data point.

Quality flags are constructed on a philosophy of two qualifiers. The first described the origin of the data and the second the quality.

Possible origins of data are:

1. Original: This entails the data value is the original value. It has not been amended by NFFS
2. Completed: This entails the original value was missing and was replaced by a non-missing value.
3. Corrected: This entails the original value was replaced with another non-missing value.

Possible qualifiers are:

4. Reliable: Data is reliable and valid
5. Doubtful: The validity of the data value is uncertain
6. Unreliable: The data value is unreliable and cannot be used.

Following this specification, the table below gives an overview of quality flag enumerations

Table D.1 Enumeration of quality flags

Enumeration	**Description**
0	Original/Reliable The data value is the original value retrieved from an external source and it successfully passes all validation criteria set.
1	Corrected/Reliable The original value was removed and corrected. Correction may be through interpolation or manual editing.
2	Completed/Reliable Original value was missing. Value has been filled in through interpolation, transformation (e.g. stage discharge) or a model.
3	Original/Doubtful Observed value retrieved from external data source. Value is valid, but marked as suspect due to soft validation limits being exceeded.
4	Corrected/Doubtful The original value was removed and corrected. However, the corrected value is doubtful due to validation limits.
5	Completed/Doubtful Original value was missing. Value has been filled in as above, but resulting value is doubtful due to limits in transformation/interpolation or input value used for transformation being doubtful.
6	Missing/Unreliable Observed value retrieved from external data source. Value is invalid due to validation limits set. Value is removed
7	Corrected/Unreliable The original value was removed and corrected. However, corrected value is unreliable and is removed.
8	Completed/Unreliable Original value was missing. Value has been filled in as above, but resulting value is unreliable and is removed.
9	Missing value in originally observed series. Note this is a special form of both Original/Unreliable and Original/Reliable.

Notes:

- No difference is made between historic and forecast data. This is not considered a quality flag. The data model of NFFS is constructed such that this difference is inherent to the data type definition.
- External sources may either be an actual external source, a forecasting module or a transformation. The convention in NFFS the definition of data series parameter types identifies the data source.

E Schema Documentation

F Module Adapter Application

This memo describes application of the module adapter to allow integration with the National Flood Forecasting System (NFFS) through use of the published interface format interchange specification.

The objective of the memo is twofold:

1. to provide a description of the preferred approach in developing module adapters, and,
2. to illustrate the approach through an example taken from the Northeast region.

The document SIS Module Adapter Specification, Version 2.4 describes the use of the general adapter in allowing modules to be run from within NFFS. The *general adapter* (this module is a part of NFFS) is configured to provide the required data (both dynamic and static) in the *published interface* format. A *module adapter* (supplied by the same supplier as the module itself) is used to translate the data from the published interface format to the native module format.

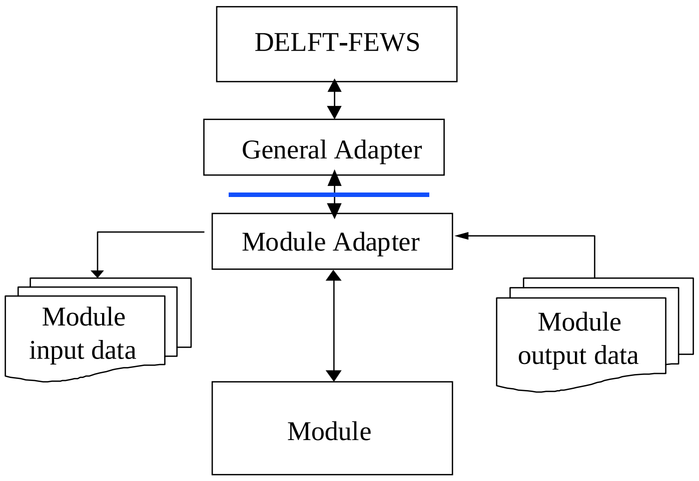


Figure 2 Schematic interaction between the General Adapter and the Module adapter through the published interface.

The general adapter is as stated an integral part of NFFS. This adapter is used to link all third party modules required to make a forecast in a configured forecast system. To allow this, each instance of the general adapter linking NFFS with a forecasting module is configured as required to provide the data inputs to the module, run the module and retrieve the data outputs from the module.

The preferred approach in running a module within NFFS is in three steps:

1. Export of data required by the module through the published interface from the NFFS database to the module native format. This data can cover dynamic time series data, parameter value sets, module states etc. A full description of the data types supported is given in SIS Module Adapter Specification, Version 2.0. To determine what data is exported to a given module, a configuration file (XML formatted) is passed to the General Adapter as an argument. This file is a part of the NFFS and is configured during set-up of a forecasting system. After the general adapter has run the input files required by the module are available in the published interface format. This can entail time series, module states etc. Figure 3 shows a schematic description of this first step. The module adapter must provide on completion an XML formatted diagnostic file giving the general adapter information on the status of the translation process. For the format of this file and associated enumeration see the published interface specification.

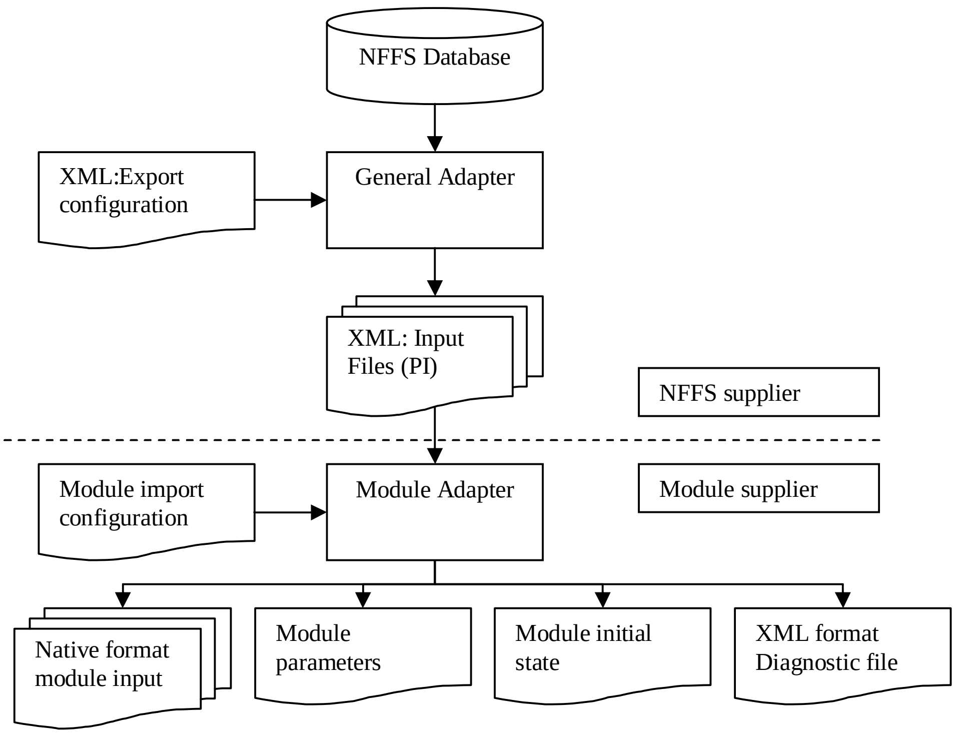


Figure 3 Detailed view of export of data from NFFS to module native format through the published Interface (PI).

2. In the second step the module itself is run. This is achieved by the general adapter executing a call to the module executable, with the possible addition of command line arguments. This module executable must be able to run in batch mode, without any user interaction required. The module executable reads the required input data, parameters and states in its native format, performs the required calculation and establishes a set of output files and states, again in the native module format. This set of output files must include a file giving diagnostic information on the module run. This file can be in the native module format. Figure 4 gives a detailed schematic overview of concept of running a module executable within NFFS.

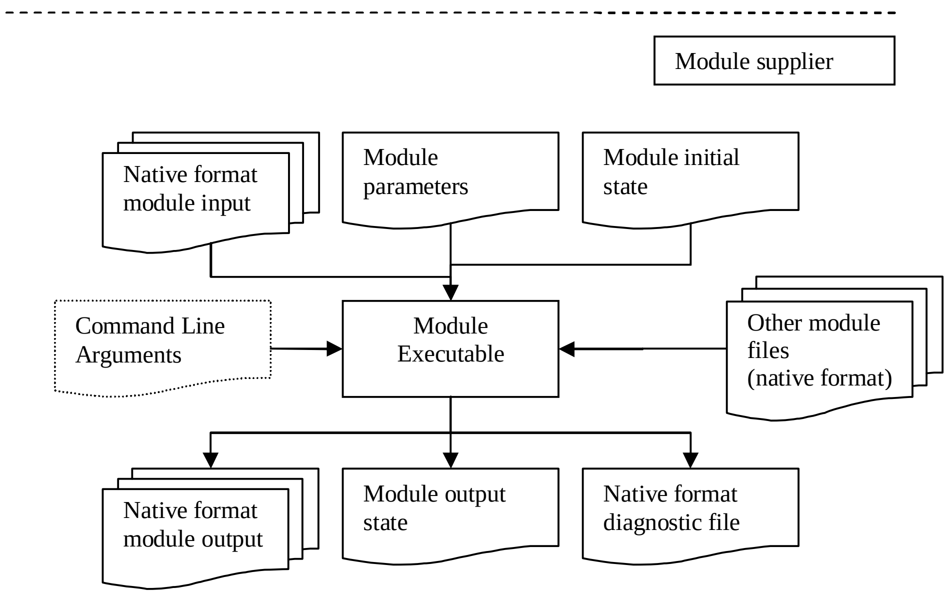

Module supplier
Native format module input
Module parameters
Module initial state
Command Line Arguments
Module Executable
Other module files (native format)
Native format module output
Module output state
Native format diagnostic file

Figure 4 Detailed view of running a module executable from NFFS.

3. The third step comprises the import of module output data into NFFS. Again this is using the published interface format. The module adapter is called to transform native module output formats to the published interface format. Once completed the output data is retrieved through the published interface format for insertion into the NFFS central database. As with the module inputs an XML formatted configuration file is used to determine what data are imported.

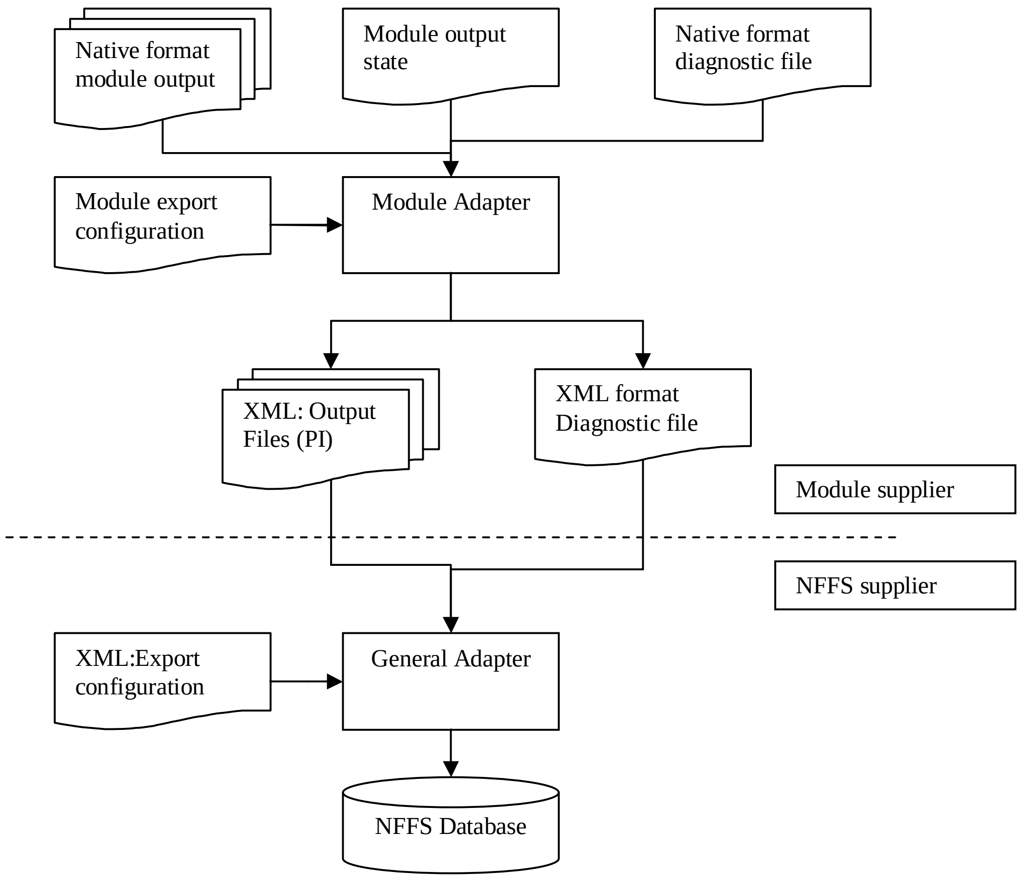


Figure 5 Detailed view of import of data to NFFS from module native format through the published Interface (PI).

The three steps make it clear that there are three functional elements to be provided by a module supplier to allow coupling of a module with NFFS. This functionality may be encompassed in either two or three separate executable.

Element	**Function**	**Configuration**	**Comment**
Module Adapter	Import data from Published Interface format to Native module format	Configuration file specifying data to import. Preferred format of file: XML	
Module executable	Run module using data in native module input format. Write output in native module output		• Batch file • Runs in dedicated work directory

	format		
Module Adapter	Exports data from native module format to published interface format	Configuration file specifying data to export. Preferred format of file: XML	

Example: Kinematic wave routing at HEBDEN bridge

As an example a comparison is made in this memo of the ICA model component file for the forecast at Hebden Bridge on the Calder River in Northeast region. Although some understanding of the structure of the ICA is helpful in this comparison, it seems that developing an example from the ICA is most appropriate given the complexity of the ICA. The full ICA module component file is included in the Appendix for reference.

Description of current approach as implemented in ICA:

Two data series are forecast for the site at Hebden Bridge. These are the water level and the discharge (described in the ICA in the forecast requirement files `lu-hebdbr1.rffs` and `qx-hebdbr1.rffs`). The discharge at the site is first calculated using the Kinematic Wave model (supplied by CEH) for the reach upstream of Hebden Bridge.

There are two inputs to the kinematic wave model, being the discharges from the Walsden sub-catchment and the Todmorden sub-catchment (ICA forecast requirements `QX-WALSDN1` and `QX-TODMDN1`).

These two inputs are routed using the KW model to form a temporary output discharge series at Hebden Bridge (Series name in ICA Model Component File: `2Q-HEBDBR1)`.

In the next step, the level series available in the database for the historic period (ICA code `LU-HEBDBR1`) is transformed to a discharge series using the applicable stage discharge curve (ICA: `RATING` algorithm). This supplies for the historic period an “observed” discharge series.

Using this “observed” discharge series, an ARMA error modelling step is carried out. This uses as input the temporary discharge calculated by the KW model and the “observed” discharge series. Using the ARMA procedure the corrected output series for Hebden Bridge is derived (ICA: QX-HEBDBR1) for the new forecast period.

In the final step levels for the forecast period are derived using this forecast discharge series again through the applicable rating curve (ICA: `RATING` algorithm).

Data series naming conventions in NFFS

In NFFS, the naming convention is a little different from that applied in the ICA. The ID’s with which series are identified are constructed on the basis of the (unique) station code. Data series available at that station are then identified by the station code plus a data type suffix. Clear distinction is made through parameter types of series available for the historic period and for the forecast period. The series ID is used as a unique identifier in locating data series. Each series does have a “name” that is used in all displays etc to enhance readability.

Description	ICA Series	NFFS series Historical	NFFS series Forecast
Input discharge series	QX-WALSDN1	H-27392-Q.hc	H-27392-Q.fc
Input discharge series	QX-TODMDN1	H-27931-Q.hc	H-27931-Q.fc
Output discharge at Hebden Bridge (simulated)	2Q-HEBDBR1	H-27932-Q.hr	H-27932-Q.fr
Output discharge at Hebden Bridge (updated)	QX-HEBDBR1	H-27932-Q.uhr	H-27932-Q.ufr
Observed Level at Hebden Bridge	LU-HEBDBR1	H-27932-h.m	Not Applicable
Forecast Level at Hebden Bridge	LU-HEBDBR1	H-27932-h.hr	H-27932-h.fr

Note: in the ICA the observed and modelled series for the historical period seem to occupy the same database location. These two are indeed the same value in case an ARMA error correction procedure is used with a parameterisation including the auto-regressive component.

NFFS Workflow for forecast at Hebden Bridge

In NFFS a series of tasks such as that required in creating the forecast at Hebden Bridge are configured in a Task File. Each of the tasks required is performed through either a standard NFFS utility (e.g. validation, interpolation, error modelling) or an external module. The external module is called through the general adapter.

The example below illustrates the workflow definition for the forecast at Hebden Bridge (all file names are as an example, and the XML structure is illustrative but not definite). The sequence of steps to deliver the forecast requirement at Hebden Bridge is the same as specified in the ICA.

Workflow files such as this are configured as a part of NFFS, not by any third party module suppliers. The first element uses the General Adapter to call the Kinematic Wave module. The configuration file is used by the General Adapter to direct how this module is used. An example of the configuration file for this element is given in the Appendix. Again this configuration file is configured during NFFS configuration, not by the third party module supplier.

```
<?xml version="1.0" encoding="UTF-8"?>

<workflow
xmlns="http://www.wldelft.nl/fews"
xmlns:target="http://www.wldelft.nl/fews"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.wldelft.nl/fews workflow.xsd" version="1.0">
        <sequence>
                <!-- Run KW for reach to Hebden Bridge (using General Adapter)-->
                <element name="kw.hebdbr">
                        <caption>KW run to Hebden Bridge"</caption>
                        <class>nl.wldelft.fews.util.GeneralAdapter</class>
                        <configuration>kw_hebdbr_historical.xml</configuration>
                        <auto>"true"</auto>
                        <output>"rating.hebdbr.observed</output>
                </element>
                <!-- Transform observed levels to discharges at Hebden Bridge
                (using ConversionServer)-->
                <element name="rating.hebdbr.observed">
                        <caption>Apply rating curve at Hebden Bridge"</caption>
                        <class>nl.wldelft.fews.util.ConversionServer</class>
                        <configuration>rating_hebdbr_observed.xml</configuration>
                        <auto>"true"</auto>
                        <input>"kw.hebdbr</input>
                        <output>"rating.hebdbr.observed</output>
                </element>
                <!-- Run ARMA_FILL through General Adapter-->
                <element name="arma.hebdbr.observed">
                        <caption>ARMA correction at Hebden Bridge"</caption>
                        <class>nl.wldelft.fews.util.GeneralAdapter</class>
                        <configuration>arma_hebdbr_historical.xml</configuration>
                        <auto>"true"</auto>
                        <input>"rating.hebdbr.observed</input>
                        <output>"arma.hebdbr.observed</output>
                </element>
                <!-- Transform predicted flows to levels at Hebden Bridge-->
                <element name="rating.hebdbr.predicted">
                        <caption>Apply rating curve at Hebden Bridge"</caption>
                        <class>nl.wldelft.fews.util.ConversionServer</class>
                        <configuration>rating_hebdbr_predicted.xml</configuration>
                        <auto>"true"</auto>
                        <input>"arma.hebdbr.observed</input>
                </element>
        </sequence>
```

```
</workflow>
```

Appendix I:
ICA Model Component file for Hebden bridge (file: CALD3-HEBDBR.RFFS)

```
CALD3-HEBDBR                                      :MOD_COMP_ID
!     Version: 1.13     Date: 9-OCT-1997 11:41
!    Revision:    2     Date:  9-OCT-1997 14:12
!     Origin : IH RFFS Model Network Set-up (Hand)
!  File-type : Model Component Description
   1                                              :MC_FILE_FORMAT
   1                                              :COMP_STORE_IND
   20961                                          :COMP_DATA_ID
HEBDEN BRIDGE   :Channel flow- kinematic wave     :COMP_NAME
   1                                              :COMP_CLASS_IND
   4                                              :COMP_POSITN_IND
   6                                              :COMP_TYPE_IND
   1   1   0                                      :COMP_PURPOSE
   0                                              :DATA_NUM_PROFI
                                                  :PROFILE_REQ_ID_PROFI
   0                                              :PROFILE_SUB_ID
   0                                              :SCALE_TYPE_PROFI
     0.00000                                      :SCALE_VALUE_PROFI
   2                                              :DATA_NUM_INPUT
QX-WALSDN1  QX-TODMDN1                            :FORECAST_REQ_ID_INPUT
   0   0                                          :SCALE_TYPE_INPUT
     0.00000     0.00000                          :SCALE_VALUE_INPUT
   0                                              :DATA_NUM_CONTR
                                                  :FORECAST_REQ_ID_CONTR
   0                                              :DATA_NUM_SPECI
                                                  :FORECAST_REQ_ID_SPECI
   2                                              :DATA_NUM_OUTPU
QX-HEBDBR1  LU-HEBDBR1                            :FORECAST_REQ_ID_OUTPU
   0   0                                          :SCALE_TYPE_OUTPUT
     0.00000     0.00000                          :SCALE_VALUE_OUTPUT
   1                                              :DATA_NUM_DUMMY
2Q-HEBDBR1                                        :DUMMY_REQ_ID
   0                                              :TIME_DELAY_IND
   3                                              :COMP_DECOMP_TYPE
   4                                              :COMP_DECOMP_NUM
FA_KW                                             :MODEL_ALGOR_ID            (1)
   2   2   0   0   0   0   0   0   0   0   0   0 :ALGOR_MSIZE_INPUT         (1)
   0   0   0   0   0   0   0   0   0   0   0   0 :ALGOR_MSIZE_CONTR         (1)
   0   0   0   0   0   0   0   0   0   0   0   0 :ALGOR_MSIZE_SPECI         (1)
   1   1   0   0   0   0   0   0   0   0   0   0 :ALGOR_MSIZE_OUTPU         (1)
   3   0   4   0   3   0   0   0   0   0   0   0 :ALGOR_MSIZE_STATE         (1)
   3   0   1   0   3   0   0   0   0   0   0   0 :ALGOR_MSIZE_TRANS         (1)
  38   1   1   3   1   4   4   2   4   4   3   0 :ALGOR_MSIZE_PARAM         (1)
  29   9   2   1   1   3   4   4   0   0   0   0 :ALGOR_MSIZE_IPARA         (1)
     9.84000     2.90000     0.00000     1.35900 :ALGOR_VALS_PARAM          (1)
     0.00000     0.00000     0.00000     1.35900 :ALGOR_VALS_PARAM          (1)
     0.00000     0.00000     0.00000     0.73300 :ALGOR_VALS_PARAM          (1)
     0.00000     0.00000     0.00000     0.73300 :ALGOR_VALS_PARAM          (1)
     0.00000     0.00000     0.00000     1.00000 :ALGOR_VALS_PARAM          (1)
     0.00000     1.00000     0.00000     1.00000 :ALGOR_VALS_PARAM          (1)
     0.00000     1.00000     0.00000     1.00000 :ALGOR_VALS_PARAM          (1)
     0.00000     0.00000     0.00000     1.00000 :ALGOR_VALS_PARAM          (1)
     0.00000     0.00000     0.00000     1.00000 :ALGOR_VALS_PARAM          (1)
     0.00000     0.00000                         :ALGOR_VALS_PARAM          (1)
   3   2   4   1   1   0   0   1   1   3   0   0 :ALGOR_VALS_IPARA          (1)
   0   1   1   0   1   2   2   0   1   3   1   0 :ALGOR_VALS_IPARA          (1)
   1   3   2   0   1                             :ALGOR_VALS_IPARA          (1)
QX-WALSDN1  QX-TODMDN1                            :DATA_STREAM_INPUT         (1)
                                                  :DATA_STREAM_CONTR         (1)
                                                  :DATA_STREAM_SPECI         (1)
2Q-HEBDBR1                                        :DATA_STREAM_OUTPU         (1)
   0                                              :TEST_SNOWFLAG_IND         (1)
   0                                              :SIMULAT_SKIP_IND          (1)
   0                                              :EXECUT_SPECI_IND          (1)
   6                                              :EMERG_WARM_UP             (1)
RATING                                            :MODEL_ALGOR_ID            (2)
   1   1   0   0   0   0   0   0   0   0   0   0 :ALGOR_MSIZE_INPUT         (2)
   0   0   0   0   0   0   0   0   0   0   0   0 :ALGOR_MSIZE_CONTR         (2)
   0   0   0   0   0   0   0   0   0   0   0   0 :ALGOR_MSIZE_SPECI         (2)
   1   1   0   0   0   0   0   0   0   0   0   0 :ALGOR_MSIZE_OUTPU         (2)
   0   0   0   0   0   0   0   0   0   0   0   0 :ALGOR_MSIZE_STATE          (2)
   0   0   0   0   0   0   0   0   0   0   0   0 :ALGOR_MSIZE_TRANS          (2)
   5   0   0   1   5   1   0   0   0   0   0   0 :ALGOR_MSIZE_PARAM          (2)
   2   2   0   0   0   0   0   0   0   0   0   0 :ALGOR_MSIZE_IPARA          (2)
    13.95000     0.06600     1.47800  9999.00000 :ALGOR_VALS_PARAM           (2)
  9999.00000                                     :ALGOR_VALS_PARAM           (2)
   1   1                                         :ALGOR_VALS_IPARA           (2)
LU-HEBDBR1                                       :DATA_STREAM_INPUT          (2)
                                                 :DATA_STREAM_CONTR          (2)
                                                 :DATA_STREAM_SPECI          (2)
QX-HEBDBR1                                       :DATA_STREAM_OUTPU          (2)
   0                                             :TEST_SNOWFLAG_IND          (2)
   0                                             :SIMULAT_SKIP_IND           (2)
   0                                             :EXECUT_SPECI_IND           (2)
   0                                             :EMERG_WARM_UP              (2)
ARMA_FILL                                        :MODEL_ALGOR_ID             (3)
   1   1   0   0   0   0   0   0   0   0   0   0 :ALGOR_MSIZE_INPUT          (3)
   0   0   0   0   0   0   0   0   0   0   0   0 :ALGOR_MSIZE_CONTR          (3)
   0   0   0   0   0   0   0   0   0   0   0   0 :ALGOR_MSIZE_SPECI          (3)
   1   1   0   0   0   0   0   0   0   0   0   0 :ALGOR_MSIZE_OUTPU          (3)
   4   0   0   1   2   2   0   0   0   0   0   0 :ALGOR_MSIZE_STATE          (3)
   0   0   0   0   0   0   0   0   0   0   0   0 :ALGOR_MSIZE_TRANS          (3)
   2   0   2   0   2   0   0   0   0   0   0   0 :ALGOR_MSIZE_PARAM          (3)
   3   3   0   0   0   0   0   0   0   0   0   0 :ALGOR_MSIZE_IPARA          (3)
    -1.50830     0.61721                         :ALGOR_VALS_PARAM           (3)
   2   0   1                                     :ALGOR_VALS_IPARA           (3)
2Q-HEBDBR1                                       :DATA_STREAM_INPUT          (3)
                                                 :DATA_STREAM_CONTR          (3)
                                                 :DATA_STREAM_SPECI          (3)
QX-HEBDBR1                                       :DATA_STREAM_OUTPU          (3)
   0                                             :TEST_SNOWFLAG_IND          (3)
   0                                             :SIMULAT_SKIP_IND           (3)
   0                                             :EXECUT_SPECI_IND           (3)
   4                                             :EMERG_WARM_UP              (3)
RATING                                           :MODEL_ALGOR_ID             (4)
   1   1   0   0   0   0   0   0   0   0   0   0 :ALGOR_MSIZE_INPUT          (4)
   0   0   0   0   0   0   0   0   0   0   0   0 :ALGOR_MSIZE_CONTR          (4)
   0   0   0   0   0   0   0   0   0   0   0   0 :ALGOR_MSIZE_SPECI          (4)
   1   1   0   0   0   0   0   0   0   0   0   0 :ALGOR_MSIZE_OUTPU          (4)
   0   0   0   0   0   0   0   0   0   0   0   0 :ALGOR_MSIZE_STATE          (4)
   0   0   0   0   0   0   0   0   0   0   0   0 :ALGOR_MSIZE_TRANS          (4)
   5   0   0   1   5   1   0   0   0   0   0   0 :ALGOR_MSIZE_PARAM          (4)
   2   2   0   0   0   0   0   0   0   0   0   0 :ALGOR_MSIZE_IPARA          (4)
    13.95000     0.06600     1.47800  9999.00000 :ALGOR_VALS_PARAM           (4)
  9999.00000                                     :ALGOR_VALS_PARAM           (4)
   2   1                                         :ALGOR_VALS_IPARA           (4)
QX-HEBDBR1                                       :DATA_STREAM_INPUT          (4)
                                                 :DATA_STREAM_CONTR          (4)
                                                 :DATA_STREAM_SPECI          (4)
LU-HEBDBR1                                       :DATA_STREAM_OUTPU          (4)
   0                                             :TEST_SNOWFLAG_IND          (4)
   0                                             :SIMULAT_SKIP_IND           (4)
   0                                             :EXECUT_SPECI_IND           (4)
   0                                             :EMERG_WARM_UP              (4)
   1                                             :CALIBRATION_TYPE
   0                                             :CALIB_EVENT_NUM
   1   1   0   0   0                             :CALIB_EVENT_BEG            (1)
   1   19999   0   0                             :CALIB_EVENT_END            (1)
   0                                             :CALIB_EVENT_WARMUP         (1)
   0                                             :CALIB_EVENT_CARRY          (1)
   1                                             :CALIB_SCALE_TYPE
   1                                             :CALIB_ORDER_TYPE
   1                                             :CALIB_WEIGHTING_TYPE
   1                                             :UNCERTAINTY_TYPE
   0                                             :UNCERT_PARAM_NUM
     0.00000                                     :UNCERT_VAL_PARAM
UPPER CALDER RECALIBRATION OCT 1997              :CALIB_NOTES                (1)
sd fitted in model                               :CALIB_NOTES                (2)
                                                 :CALIB_NOTES                (3)
                                                 :CALIB_NOTES                (4)
                                                 :CALIB_NOTES                (5)
recalib R.AUSTIN IH                              :COMP_GEN_COMMENT           (1)
                                                 :COMP_GEN_COMMENT           (2)
                                                 :COMP_GEN_COMMENT           (3)
                                                 :COMP_GEN_COMMENT           (4)
```

```
                                                    :COMP_GEN_COMMENT           (5)
```

Appendix II:
Configuration file for running KW module at Hebden Bridge using General Adapter

```
<?xml version="1.0" encoding="UTF-8"?>

<dyntasks
xmlns="http://www.wldelft.nl/fews"
xmlns:target="http://www.wldelft.nl/fews"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.wldelft.nl/fews dyntasks.xsd" version="1.0">
        <general exportpath="file: nffs/kw/input" importpath="file: nffs/kw/output"/>
        <sequence>
                <!-- export time series for discharge for Todmorden-->
                <export seriesid="H-27931-Q.hc" file="file:todmdn.xml">
                        <xml format="fews"/>
                </export>
                <!-- export time series for discharge for Walsdn-->
                <export seriesid="H-27392-Q.hc" file="file:walsdn.xml">
                        <xml format="fews"/>
                </export>
                <!-- export parameters for KW model to Hebden Bridge-->
                <exportparams>
                        <moduleid>kw.hebdbr</moduleid>
                        <paramsetid>kw.hebdbr.par</paramsetid>
                        <filename>file:./kw_hebdbr_par.xml</filename>
                </exportparams>
                <!-- export module initial state -->
                <writestate stateid="kw.hebdbr.state"/>
                <!-- start kw adapter to transfrom XML data to KW format -->
                <!-- note:all settings are examples of filenames. it is assumed the kw
                module adapter requires one argument. This is an XML file with the
                relevant configuration! -->
                <task diagfile="file:./kwadapter_diag.xml" moduleid="kw.hebdbr">
                        <deletefile>file:kw_adapter.rtn</deletefile>
                        <exe>file:nffs/bin/kw_adapter.exe</exe>
                        <workdir>nffs/kw/workdir</workdir>
                        <arg>nffs/kw/config/kw_hebdbr_input.xml</arg>
                        <taskreturn>file:kw_adapter.rtn</taskreturn>
                        <taskfail>NONEXISTS</taskfail>
                </task>
                <!-- start kw module -->
                <task diagfile="file:kwmodule_diag.xml" moduleid="kw.hebdbr">
                        <deletefile>file:kw_exe.rtn</deletefile>
                        <exe>file:nffs/bin/kw.exe</exe>
                        <workdir>nffs/kw/workdir</workdir>
                        <arg>nffs/kw/config/kw_hebdbr.rffs</arg>
                        <taskreturn>file:kw_exe.rtn</taskreturn>
                        <taskfail>NONEXISTS</taskfail>
                </task>
                <!-- Start kw adapter to import data form native to XML -->
                <task diagfile="file:kwadapter_diag.xml" moduleid="kw.hebdbr">
                        <deletefile>file:kw_adapter.rtn</deletefile>
                        <exe>file:nffs/bin/kw_adapter.exe</exe>
                        <workdir>nffs/kw/workdir</workdir>
                        <arg>nffs/kw/config/kw_hebdbr_output.xml</arg>
                        <taskreturn>file:kw_adapter.rtn</taskreturn>
                        <taskfail>NONEXISTS</taskfail>
                </task>
                <!-- importing of timeseries -->
                <import seriesid="H-27932-Q.hr" file="file:kw_hebdbr_q.xml">
                        <xml format="fews"/>
                </import>
                <!-- import resulting module state -->
                <readstate stateid="kw.hebdbr.state" statename="state kw to hebdbr">
                        <stateloc type="file">
                                <readlocation>nffs/kw/state</readlocation>
                                <writelocation>nffs/kw/state</writelocation>
                        </stateloc>
                </readstate>
        </sequence>
</dyntasks>
```

Appendix III:
Example of an output XML file (shown in part)

Filename: KW_HEBDBR_Q.XML

```
<?xml version="1.0" encoding="UTF-8"?>

<timeseries xmlns="http://www.wldelft.nl/fews"
xmlns:target="http://www.wldelft.nl/fews"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.wldelft.nl/fews
timeser.xsd" version="1.0">
        <header>
                <type>instantanious</type>
                <timeseriesid>kw.output.1</timeseriesid>
                <content>Discharge</content>
                <timestep>3600</timestep>
                <missval>-999</missval>
                <longname>Simulated discharge at Hebden Bridge</longname>
                <stationName>Hebden Bridge</stationName>
                <units>m3/s</units>
                <locationid>H-27932</locationid>
                <sourceOrganisation>Northeast Region</sourceOrganisation>
                <sourceSystem>KW module</sourceSystem>
                <fileDescription>XML Data</fileDescription>
                <creationDate>2003-05-05</creationDate>
                <creationTime>14:42:14</creationTime>
        </header>
        <event date="2003-01-07" time="08:00:00      " value="2.480000" flag="0"/>
        <event date="2003-01-07" time="09:00:00      " value="2.470000" flag="0"/>
        <event date="2003-01-07" time="10:00:00      " value="2.460000" flag="0"/>
        <event date="2003-01-07" time="11:00:00      " value="2.440000" flag="0"/>
        <event date="2003-01-07" time="12:00:00      " value="2.420000" flag="0"/>
        …
        …
        <event date="2003-01-17" time="02:00:00      " value="1.490000" flag="0"/>
        <event date="2003-01-17" time="03:00:00      " value="1.480000" flag="0"/>
        <event date="2003-01-17" time="04:00:00      " value="1.480000" flag="0"/>
        <event date="2003-01-17" time="05:00:00      " value="1.480000" flag="0"/>
        <event date="2003-01-17" time="06:00:00      " value="1.480000" flag="0"/>
        <event date="2003-01-17" time="07:00:00      " value="1.490000" flag="0"/>
</timeseries>
```

G Questions and aswers

G.1.1 The interface describes an on-line or operational system. However the XML templates and data types described apply to off-line modelling. Is it the intention for the DELFT-FEWS databases to hold data such as cross-sections, river channels etc. which are used to develop the model from scratch?

The interface is indeed intended to be operated an on-line operational system. It's prime communication with external forecasting modules is through dynamic data such as time series, model states and diagnostics (see below). The published interface formats have been defined to cover a wider range of data types, but these need not be used in all cases – i.e. there is no requirement to allow communication of all data types. Indeed there is currently no module that caters for the full range of formats. Although the EA is to decide to what extent data should be passed, the practical approach followed with e.g. ISIS and the CEH modules is to pass only time series, module states and diagnostics information.

In exchanging data between a module and NFFS, a prioritisation of data types can be identified.

Priority	Data Type	Comment
1	Time Series Module states Diagnostics	sufficient to cover most modules used in flood forecasting systems, including e.g. Mike-11 and NAM
1 (a)	Time series of grid data	additionaly required for modules with 2D I/O formats (e.g inundation codes)
2	Parameters	Module parameters may be passed where the module allows calibration through the NFFS calibration facilities
2 (a)	Longitudinal profile data	For hydrodynamic modules longitudinal data types may be passed – not a strict requirement.
3	Static data (cross sections, branches, etc)	This data is not required for operational forecasting. In exceptional cases data such as on branches may be required (for display purposes), but this need not be passed through the adapter and can be configured as appropriate.

G.1.2 Will the HarmonIT developments be used in the module adapter? If so, will appropriate tools be provided?

In its current form the General Adapter does not use the HarmonIT (OpenMI) module interchange facilities. The DELFT-FEWS application will in the near future be extended with an OpenMI compatible adapter, but this is as yet not considered an NFFS requirement.

G.1.3 A major part of the document consists of XML templates for a wide range of data types. It is our understanding that relatively few of these are used in practice and that these are simply made available to ensure a general system.

This assumption is correct – see also G.1.1

G.1.4 On what platform is the system running (Windows, Linux or other)?

The DELFT-FEWS system has been developed in JAVA and is platform independent. However, in the configuration of NFFS, all runs of forecasting modules are executed on dedicated servers. Presently, these are Windows systems.

G.1.5 What is the module adapter (script, bat-file, win32dll, win exe, or other)? There are no technology specifications for the adapter.

There is no specific requirement for the module adapter. The only requirement is that it can communicate through the XML file formats. The use of standard methods for reading and writing XML files, based on the schemas provided is highly recommended. This not only reduces implementation efforts but also guarantees compatibility. The general adapter currently allows the external module to be run either as an executable, or as a Java method. A minor extension will allow running of DLL's. Batch files are not recommended as the General Adapters monitors return codes from the external module. These are not always correctly passed in Batch files, and as such ungraceful failure of a module is not easily identified.

G.1.6 How does the module adapter and the general adapter communicate (win32 API, files, or other)?

The general adapter initiates the executable, Java method (or DLL) through the system/Java calls. Arguments may be passed when required, though these must be static. The general adapter may also be used to set environment variables for the module where required.

The module normally runs in a dedicated directory (tree). a working directory can be configured to act as the root of the module run.

G.1.7 Figure 1 on page 4-1 indicates that all communication between the NFFS system and the system that is delivered by a model supplier goes through the Module adapter. I this a correct assumption and can everything under the blue line be considered as a black box ?. If so, is it possible to describe the specific requirements to such a black box ?

This is correct. The most specific requirement is that the module must not have any manual interaction. This would preclude running the module in a distributed system.

G.1.8 Are there any specific requirements for the data formats used in communication between the module and the module adapter (e.e. binary or ASCII?)

This is a sole choice of the module adapter. There is a slight preference to ASCII formats as this eases debugging. However, if this has implications as to module performance (additional pre-processing requirements) then the binary format is advocated.

G.1.9 Could we ask for further clarification to distinguish between module states and module results and under what circumstances these are stored in the NFFS databases?

The general adapter has full state management facilities. This means the module is provided with an initial state for the start of run. The module should also provide a resulting state at the end of run, or at an intermediate time. When required this resulting state is administered and stored in the NFFS database for later use.

It is important to note that the actual module state file is handled blindly - ie it is not interpreted. NFFS takes the state and time-stamps it for storage in the database. NFFS will not change the content of the state, though renaming of files/directories on import/export is possible.

For cold start runs a default state is provided to the module.

Module results are generally passed back through the Published Interface XML format and stored in the database. Obviously only the required locations need be passed.

G.1.10 What tools are provided within the system for derived data such as catchment mean rainfall, catchment evaporation, etc.?

DELFT-FEWS provides a full set of generic tools for these derived data, including all of the above. Generally DELFT-FEWS is configured to pre-process all data and provide the module directly with required inputs. The module need not do any additional processing.

G.1.11 What tools are provided for updating/data assimilation and how are these integrated into the module structure shown in figure 1?

DELFT-FEWS provides an internal ARMA error modelling tool. This is module independent and is run in sequence to the module when required.

G.1.12 What adapter mechanisms will be used for sequences of modules?

Sequences of modules and data handling functions are run in sequence by so-called workflows. This means that a workflow may be configured to first run a rainfall-runoff model (e.g. NAM, PDM) through the General adapter, then run an error correction routine to the output and subsequently run a hydrodynamic module (e.g. ISIS, MIKE11, SOBEK)

G.1.13 What are the requirement for migrating from an stand-alone system to an on-line system for the module adapter

There is no difference to the module if it is on or off line. This layer is managed by DELFT-FEWS.

G.1.14 Diagnostics: must all diagnostics from a model run be reported in one diagnostics-file or does the interface allow for multiple files (this could be relevant if the model adapter executes a sequence of sub-modules during a model-run)?

In its current form the diagnostics file to each external module run should be one file. The general adapter does, however, allow for executing a sequence of modules, with a diagnostics file for each.

G.1.15 Diagnostics: can any information be passed back – or is it limited to the status type messages with one line of text per status code?

The diagnostics passed back should be relatively limited – there is no strict requirement on the length of the message, but to the point messaging is advocated. The message should provide the level of knowledge to the operational forecaster to monitor the system. Different levels of messages should be used, with information useful in debugging being labelled appropriately.

For more detailed messaging – native module formats can be used, but this is only for specialist analysis. It should be noted that when a module fails, a zipped dump files is made of all module files and I/O. This is set aside for later, detailed, analysis.

G.1.16 Execution: since all model runs of a particular setup seems to be running in the same fixed location (URI) it is the responsibility of the run-time adapter to clean up used files after a model run. Will DELFT-FEWS overwrite existing files during subsequent runs (e.g. the input TS XML files)? This would reduce the cleaning up to files, which are not overwritten by the model tool itself.

It is good practice for a module to clean up redundant files after use. The general adapter does allow configuration of a purge activity either before the module run or after the module run (or both).

G.1.17 Dynamic files: who decides the location of the dynamic files which are created by the general adapter? Both the model adapter and the general adapter must know the location (one to write and the other to read them), but who determines the structure in which they are created?

In the general adapter configuration (XML) the location of the PI-XML files to pass to the module is given, as well as the location where PI-XML files are expected. The location of diagnostics files and work directories can also be configured. The structure is dictated by the Published Interface. All files passed to the module will be in the same directory and all files passed back are expected in the same directory (with the exception of the diagnostics file).

G.1.18 Dynamic files: Can/will DELFT-FEWS pass other dynamic module parameters? Especially, how will DELFT-FEWS pass information of simulation period – start date/time and end date/time and optionally time-of-forecast date/time for the model module to use?

To ensure simplicity, the length of the module run is dictated by the length of the time series passed. When a distinction is to be made between forecast and historic run, these will be passed as separate time series. The module adapter should identify run times from these.

G.1.19 Model states: It is assumed that DELFT-FEWS decides which of the available states (default, scenario) will be supplied to. How will the model adapter know which type of model state to return?

Indeed. The module state returned will be administered by DELFT-FEWS. Depending on the status of the run it will be administered as a state to be used in ensuing forecasts, or disregarded.

G.1.20 We presume that the adapter will run at the time of forecast, and therefore the general adapter will provide data on request for specified intervals. Correct?

Indeed. The General Adapter is configured to provide the module with the required data. Data may be provided at equidistant or non-equidistant intervals.

G.1.21 Is it correct that one module run can supply one set of state information as output, i.e. data for one timestamp (and time step) only?

Yes. A state represents the state of a module at a single point in time.

G.1.22 Please advise on what quality checks are made for observations used for updating that can be used to switch off updating if the updating observations are missing or of poor quality.

The following quality checks can be made within DELFT-FEWS (when configured by the user)

- Detection of outliers (data flagged unreliable)
- Check of exceedance of hard limits (data flagged unreliable)
- Check of exceedance of soft limits (data flagged doubtful)
- Rate of change check. When a rate of change exceeds a pre-set value, data is flagged unreliable
- Same readings check: When data remains within a pre-set range for a configured period, data is flagged unreliable
- Temporary shift check. When a ‘temporary shift’ is detected, data is flagged unreliable

G.1.23 Can paths for files and directories include several levels of subdirectories - or only one relative to the general paths?

Several levels are possible.