WebDC3 Web Interface Documentation

Release 1.0.3

M. Bianchi, P. Evans, A. Heinloo, J. Quinteros

October 11, 2013

CONTENTS

CHAPTER

ONE

THE WEBDC3 WEB INTERFACE GENERATOR

This code was developed from the old webdc.eu portal developed at GFZ in the NERIES project. In the first stage we decided to maintain the functionalities already achieved focusing on a code clean-up and technology upgrade to accommodate the current EIDA needs.

The new web interface looks different, but functions more or less like the old one. Users can select waveforms, dataless SEED, and inventory XML for downloading. The selection can be constrained by streams by network, station location, channel and other properties, and the time windows chosen can be constrained based on user-selected events.

The web interface mainly uses JavaScript for presentation, with Python used to provide underlying services.

This documentation contains:

- A *User Guide* for getting data from the running web interface.
- *Operator Instructions*, for installing and configuring the software.
- The *Developer Notes*, for understanding the internal functions and contributing new code such as event services.

As an appendix, there is a *Self-study Tutorial* as a base to get your users familiar with what they can do with the tool.

We hope you find it useful.

CHAPTER

TWO

USER GUIDE

2.1 Introduction

The WebDC3 web interface is primarily a tool for obtaining seismic waveforms. As the name suggests, it offers an easy interactive point-and-click interface which is convenient for when you are exploring the available data, or for smaller requests. But it can be used in a few interesting additional ways too.

There are a couple of ways to use WebDC3:

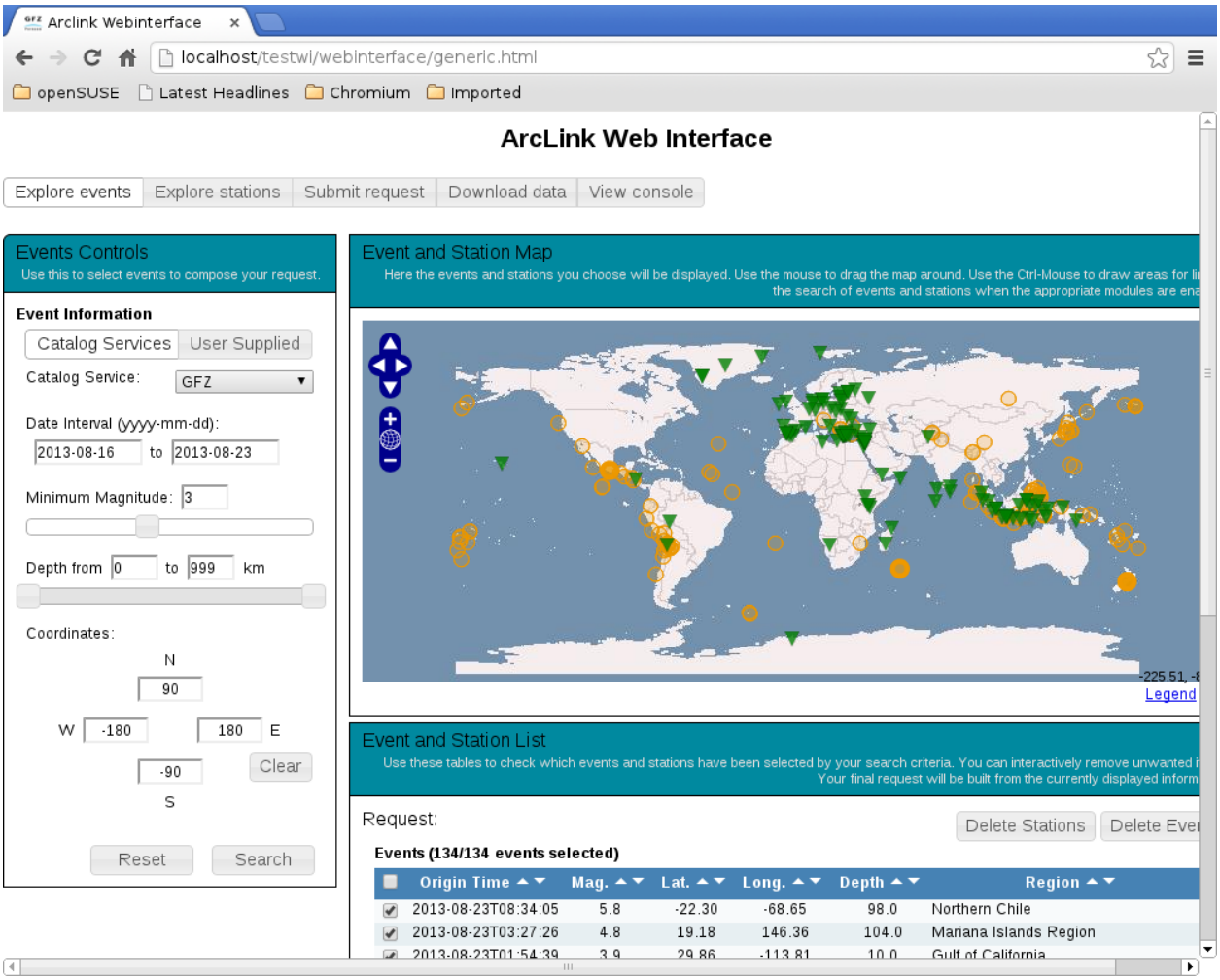
1. Event based - for exploring a catalog of seismic events (earthquakes), or for when you are looking for waveforms recorded near the time of one or more specific events. You can select events by multiple criteria, then pick from channels available at those times.
2. Station-based - to explore inventory to see what stations/streams are available and their parameters.
3. Time-span based - e.g. for obtaining station metadata over fixed periods of interest.
4. To examine the status of your requests.

There is some on-line help available as pop-ups in the [?] box at the top right of each box in the web interface. Clicking on this takes you to the appropriate part of the help page. Also you can click on the link in the top right corner to see the whole help page.

Note: The web interface is highly configurable. Your site operator may customize its appearance in many different ways. The instructions here are written with the GFZ interface in mind, but the basic work flow described below should be applicable to most sites' implementations.

2.2 Getting started

Visit http://eida.gfz-potsdam.de/webdc3 or your local webinterface site. The screen should look something like this:



There are different areas visible. Most prominent is the world map which will show stations and events as they are selected. On the left are different "control" boxes to pick stations and events, and submit your data request when you are ready. Below the map is status information and a summary of the stations and events you have selected.

2.3 Event-based search

First pick a catalog from the pull-down menu in the "Select stations" area. By default the time period is 7 days Move the start date back a few days. Pick the 'GFZ' catalog, and you will see some selectors to constrain your choice of events below the time period area. You can use this to restrict your selection to a particular magnitude range e.g. greater than 5.0. There are selectors for:

- magnitude
- depth
- region - rectangular (today) and circular (planned)

in addition to period of interest. When you are happy, press 'Submit" in the "Select stations". Now in the "Display event/station" area you should see a list of events. They are also shown as circular features on the map area. These can be sorted by different criteria (triangle symbols on the top of the table), and selected/deselected as you wish.

2.3.1 Uploading a catalog of events

In addition to the public catalogs you can supply your own list of events. In the "Events Controls" box:

1. Choose "User Supplied".
2. Click "Upload Catalog". The "Catalog Input Dialog" box appears.
3. Specify what columns your data is in. You need only provide location (latitude, longitude, depth) and event time.
4. Paste your catalog data into the text area provided. It can be much longer than fits in the box.

Note: The parser attempts to determine what separator and quoting conventions were used for your input, and accommodate them, but this is not always effective. If there were problems, they should be reported on the console (click the "View console" tab at the top of the page). If possible, use comma (',') as a separator, and quote any text strings, **including the date-time string**.

Note: The parser attempts to be lenient in interpreting your data. But date-times should be like:

```
YYYY-mm-ddTHH:MM:SS
```

or you *will* run into problems. Also to avoid confusing the sniffer, make sure your data is consistently formatted from one row to the next. The parser also accepts a header row. If one or more rows in your CSV data are unacceptable, they will be ignored, silently. In this case you will see zero events in the Event and Station List box.

---

5. Click "Send" in the "Catalog Input Dialog" box. There is an acknowledgment pop-up. *Maybe: click "Search" in the "Events Controls" box to load your submission.* If parsing was successful, you will see your events on the map and in the "Event and Station List".

2.4 Stations/channels search

In the "Explore stations" tab you are able to explore and select the available stations and channels. There are different possibilities to filter the stations and channels. To follow them in top-bottom order (as they appear on the web page) is recommended, but it is not mandatory.

Start by choosing a time range in years with the double slider at the top. The default values are 1980 [1] and the current year, which covers the whole range of operation. When you change the time selection, the drop-down lists are updated to show the available information only for this time range.

You can then refine by specifying the network type, and/or a particular network.

When you want to select the stations there are three different ways to do it.

1. by **station code**: You can use the drop-down list to select one particular station or all of them.
2. by **geographic region**: You can enter the minimum and maximum latitude and longitude to define a rectangular area. In this case, all the stations located inside this area, *and that also meet the other selection criteria*, will be selected. The rectangular area can also be selected in the map, by pressing the left "Shift" and dragging the mouse over the map.
3. by **events**: If you have already selected at least one event (it should be visible on the list under the map) you can select stations located within a certain distance (in degrees) and azimuth of an event.

[1] Remember, the web interface sits on top of Arclink, and Arclink inventory generally begins on 1 January 1980.

To further select/filter the desired streams you have two options:

1. by **code**: Just click on the list of streams you would like to request. You can also used the "Shift" and "Ctrl" keys to make multiple selections.
2. by **sampling rate**: With the slider control, select the preferred sample rate that you want to get from the station. The web interface will return the channels which are closest to the preferred sampling rate. This means that *at least* one channel will be retrieved per station.

Once the filter criteria are entered, you can click on "Search" and the resulting list of stations/channels will appear in the list below the map.

After you have made one selection, you may append additional stations (use the "Append" button, where the "Search" button was before you made a selection). Or you may replace your selection using the "Delete Stations" button on the "Event and Station List".

`[BUG, October 2013: Appending extra streams to an existing set of stations doesn't work!]`

2.4.1 Further filtering

If you take a look at the top of the "Stations list" you will see a small "Filter" button on the right part. When you click on it, you are presented a summary of the available Location, Sampling, Instrument and Orientation Code. By default, everything is checked and you can use these check boxes to further filter the channels you want in your request. For instance, if in "Orientation Code" you left just "Z" checked you will include only the channel associated with the vertical component.

Note: Remember that you need to click again on "Filter" for your changes to take place.

You can also use the check boxes at the left of every line (station) to select all the stations that you want and click on "Freeze" to remove all the unchecked stations from the list.

When you finished selecting all the information related to events and stations you can go to the "Make Request" control using the "Submit Request" tab.

2.5 Request types

There are two different types of information that you can get from this system:

- waveform data: there are two formats in which you can download, mini-SEED and full SEED.
- inventory metadata: there are also two formats in which you can download the information, dataless SEED and ArcLink Inventory XML.

In order to be able to create *any* type of request you need to have at least one channel selected.

2.6 Making a request

On the "Submit Request" tab, you must first select the request type. You may enable bzip2 compression. Compression is recommended for text-based formats like dataless SEED and XML. In the case of dataless and full SEED, you can elect to use a response dictionary; this makes SEED metadata of some networks substantially smaller, but may cause compatibility problems.

Next you can select an absolute or relative (to P and S waves) time window. If you haven't selected any events, then the absolute mode is the only choice, otherwise you almost certainly want to use the relative mode.

Finally click "Review" or "Submit". "Review" opens an additional pop-up window, where further adjustments to the final request can be made. Clicking "Submit" skips this review step.

At this point, it is checked whether the request size is within configured limits. If the check is passed, multiple Arclink requests are created and routed to different data centers. WebDC refers to this set of Arclink requests corresponding to a single submit action as a "request group".

2.7 Status/download

On the "Download data" tab, in the "Recent Requests" box, you should now see a line corresponding to the request group created during the previous step. Once routing is complete, you can click on the line to open a pop-up showing the status of the request group.

Sometimes copies of data are stored in multiple data centers; in this case there are multiple routes to the data. If the first route returns no data, it is possible to reroute the request to the next data center.

The following buttons are attached to each request group:

- Reroute: tries to send all lines with NODATA and RETRY status to alternative data centers if possible. If there are no (more) alternative routes, you'll see "No more routes found" on the console.
- Retry: same as Reroute, except that lines with RETRY status are sent to the same data center again.
- Resend: send the same request group again. This might be helpful if there are transient errors. Note that the re-sent request does not include lines which could not be routed originally because no routes were found (those lines are not part of the request group).
- Delete: deletes the request group in all data centers involved.
- Refresh: contacts the server(s) to update the processing status of the request group. If you click here during a big request, you will likely see the number of "PROCESSING" lines increase and the number of "UNSET" lines decrease.

In the "Manage Requests" box, you can display the status of all requests associated with your user ID (currently, e-mail address) in all EIDA data centers. Here you also have the option of downloading all data volumes with a single click if you have jDownloader running.

Note: You can get jDownloader from <http://jdownloader.org/>. We recommend that you avoid the Windows exe installer and to use the MULTIOS zip instead. You can execute the jar file directly using "java -jar -Xmx512m JDownloader.jar".

2.8 Limitations

Using WebDC3 you can generate requests which involve many time windows for many streams/channels. These large requests may be rejected by the underlying Arclink server. In this case you will see an alert box.

At GFZ, the current limits are

- 500 events
- 10000 total request lines (traces)

The web interface can break large events up into chunks, but it is still possible for very large requests to exceed limits.

CHAPTER

THREE

MODULES

3.1 Arclink

3.1.1 webinterface

The webinterface module is an Arclink client for retrieving waveforms based on events or inventory.

Description

Configuration options for webinterface

This refers specificially to the application providing web services. See also JavaScript documentation elsewhere??

Note: This file is to be used with webinterface.xml for magical configuration documentation generation. Much more to do.

Configuration

Note: webinterface is a standalone module and does not inherit *global options*.

```
etc/defaults/webinterface.cfg
etc/webinterface.cfg
~/.seiscomp3/webinterface.cfg
```

Note: **arclink.*** *Parameters relating to connections to the Arclink server(s) supporting webinterface.*

`arclink.address`
Type: *string*

The server to connect to, given as hostname:port Default is `eida.gfz-potsdam.de:18002`.

arclink.timeout.request
Type: *int*

Timeout, in seconds, for XXX requests? Default is `300`.

arclink.timeout.status
Type: *int*

Default is `300`.

arclink.timeout.download
Type: *int*

Default is `300`.

arclink.timeout.networkXML
Type: *string*

Default is `eida.xml`.

Note: event.* *Parameters relating to event look-up services.*

---

event.defaultLimit
Type: *int*

Maximum number of events which are returned if no other limit is set. Default is `500`.

event.verbosity
Type: *int*

Verbosity level a la SeisComP logging.level. How chatty should event service be? This specifies logging on the server. 0:quiet, 1:error, 2:warning, 3:info, 4:debug. Default is `2`.

event.catalogs.ids
Type: *list*

List of all event services which will be enabled. Include 'parser' here to support file upload. For each service, there must be an EventService object with a handler method suitable for the service. There may be more than one instance of each EventService with different configuration parameters (e.g. different baseURL), in which case they are distinguished by the names here. Default is `geofon, comcat, emsc, parser`.

event.catalogs.preferred
Type: *string*

The preferred service appears first in the pull-down menu of event services. Default is `geofon`.

event.catalogs.registeredOnly
Type: *boolean*

NOT NEEDED?? If this parameter is 'true', any service, e.g. {id}, with no description will be *hidden*. That is, it will be registered, so requests to "/event/{id}" will be handled, but it will not be listed in the "Events Services" dialog box. Only those services with a non-empty "description" attribute will be displayed. Default is `True`.

event.names.lookupIfEmpty
Type: *boolean*

If an event service does not provide a region name (e.g. an F.-E. region) should we look it up? Default is `True`.

event.names.lookupIfGiven
Type: *boolean*

If an event service *does* provide a region name (e.g. an F.-E. region) should we look it up anyway, and override what was supplied up? Default is `False`.

`event.service.description`
Type: *string*

Short description for this target event service. This is provided in the catalog (/event/catalogs). It is displayed in the presentation layer under XXX.

`event.service.baseURL`
Type: *string*

URL of the target event service. In forming queries of service {name}, baseURL is followed by '?' and any parameters, followed by event.service.{name}.extraParams

`event.service.extraParams`
Type: *string*

Any additional string which needs appending to URLs for queries of the target event service. In forming queries of service {name}, extraParams is added to form the URL `event.service{name}.baseURL?{request parameters}\&{extraParams}`

Note: js.* *Parameters relating to JavaScript. These are read in by the Python WSGI and served to the browser when ...*

`js.maptype`
Type: *string*

Type of OpenLayers map to produce. Can be one of: wms, google (in the future OSM). Default is `wms`.

Note: js.google.* *Type of Google map*

`js.google.layer`
Type: *string*

Parameter for Google Default is `Google Physical`.

Note: js.wms.* *Type of WMS map*

`js.wms.server`
Type: *string*

URL, location of the WMS map server

`js.wms.layer`
Type: *string*

Parameter for WMS

Note: js.events.* *These parameters are used in forming the Event Search Control.*

`js.events.date_startoffset`
Type: *string*

Relative time for setting the default 'start' field in the Event Search Control. Use *d*, *w*, *m*, *y* for days, weeks, months or years; this value is interpreted by the jQueryUI datepicker . Default is `-7d`.

js.events.magnitudes_minimum
Type: *float*

Default lower magnitude limit for events Default is `3.0`.

js.events.depth_minimum
Type: *float*

Default smallest depth limit (in kilometres) for events. Default is `0`.

js.events.depth_maximum
Type: *float*

Default largest depth limit (in kilometres) for events. Default is `1000`.

js.events.coordinates_north
Type: *float*

Default northern latitude limit (in degrees) for rectangular region search. Default is `90`.

js.events.coordinates_south
Type: *float*

Default southern latitude limit (in degrees) for rectangular region search. Default is `-90`.

js.events.coordinates_west
Type: *float*

Default western longitude limit (in degrees) for rectangular region search. Default is `-180`.

js.events.coordinates_east
Type: *float*

Default eastern longitude limit (in degrees) for rectangular region search. Default is `180`.

CHAPTER

FOUR

OPERATOR INSTRUCTIONS

4.1 WebDC3 web interface generator

Here we outline what you may need to do to get the web interface up and running on your site. Things may be different for your site depending on your operating system, web server, network policies and so on.

The web interface mainly uses JavaScript for presentation, with Python used to provide underlying services.

Note: WebDC3 has a modular design. Here goes something about the modules:

- presentation
- events
- stations
- requests
- maps
- console

See the *Developer Notes* for more details about the modules.

4.1.1 Presentation

The design adopted for the implementation uses Ajax queries to load the individual page blocks. The final page assembles those blocks. You (a web site operator) have complete freedom to build your own page layout from the basic supplied blocks. The basic blocks are:

1. Event Search Control block
2. Station Search Control block
3. Mapping Control block (plot events and stations)
4. Request Control block
5. Submitting block
6. Status Control block
7. Status Results block
8. Console block

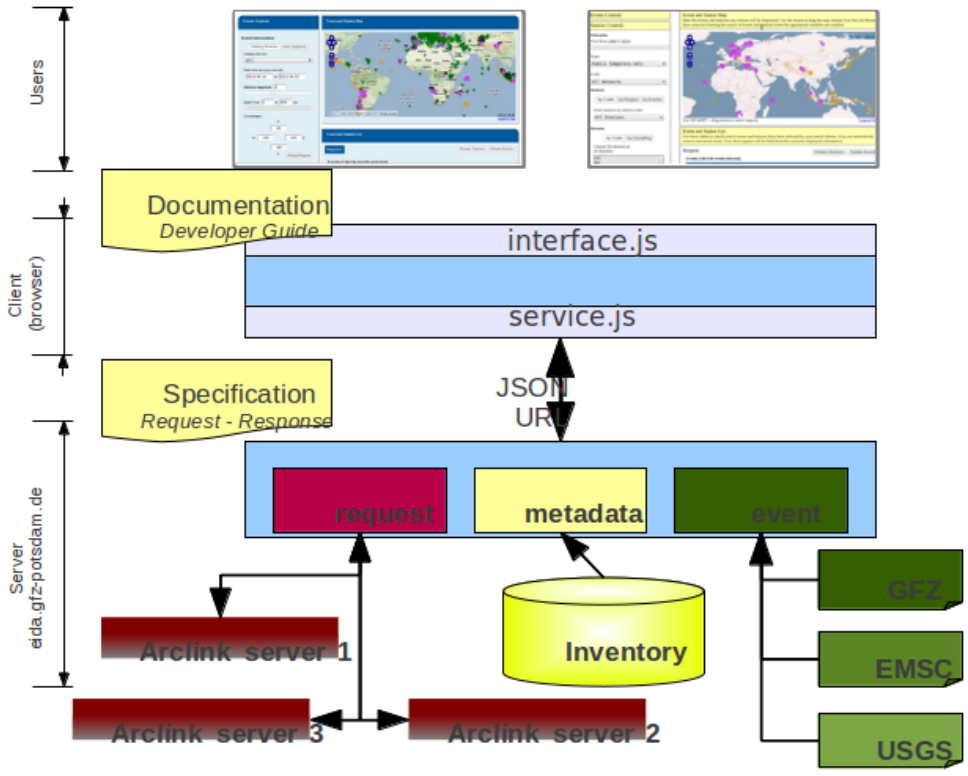
Further blocks can be implemented and later integrated into the current architecture design. Since each block is a self-contained unit we believe they will fit pretty well in any CMS or existing web pages at EIDA nodes, or even the EIDA portal at ODC.

4.2 Python and JavaScript (JS)

The complete interface needs a Python back end running, using the WSGI interface. In the Apache web server this is implemented in mod_wsgi. The back end uses the SeisComP seiscomp3 Python libraries for distance and travel time computation, configuration, and logging. The Python back end is responsible for:

1. Fetch events information for the presentation layer (JavaScript) from different web services or databases.
2. Fetch NSLC (network-station-location-channel, i.e. inventory) information to the presentation layer from an Arclink server.
3. Place requests to one or many Arclink servers.
4. Send e-mail to the user about her/his requests. **FIXME: Do we still do that? Should we**
5. Fetch the status information from the Arclink server and send it to the presentation layer.

and a JavaScript set of modules that will contact the back end services and render the page on the user's browser client and control the work flows on the page.



Also built in the back end there is an option to send configuration variables as key-value pairs from the server back end to the client JavaScript layer. These variables are initially defined in a configuration file on the server. They are fetched by the JavaScript layer and any module on the client has access to those variables that helps to guide the JavaScript in rendering the page. One clear example of such variables is the Web Mapping Service (WMS) address (URL) that is

used by the Mapping control block. For the event control the default values for the magnitude filters and depth filters are also obtained from the server through this mechanism. (See *Configuration options for webinterface* for details.)

4.3 Basic Page Set

Together with this package we are also supplying a set of static pages (HTML files) that can be used as a reference on how to use the package to build your customized interface. During the development of the modules we try not to force any possible work flow. See the content of the *examples* directory.

The basic page set is composed of two pages, one for making requests and one for checking status information. The request page use demonstrate how to use the modules numbered as 1, 2, 3, 4, 5 and 6 (block list above) and the status page uses blocks numbers 6 and 7.

Note: Make a table. Add screen shots. **TODO**

The customization of the pages should be done completely in HTML, using the 'class' and 'id' attributes of HTML entities. The basic idea is that each block renders itself inside a certain ' ' element, identified by a special 'id'. Also some blocks can accept options that are passed through the 'class' attribute on the ' '. For example, the apparently empty element:

```
<div id='wi-StationSearchControl'></div>
```

would in the end be filled by the StationSearchControl block. And code like this:

```
<div id='wi-StationSearchControl' class='nosensor'></div>
```

would load the the StationSearchControl block, but the class `nosensor` would inhibit the sensor selection dialog to be present allowing each node to further customize its interface.

Note: Adding *class='nosensor'* doesn't work, Aug 31 2013.

Also, since HTML allows multiple classes to the same container options related to formatting and option for the block control would coexist on the same ' '. Furthermore on the customization part of the operator manual [REF] you can find all the special 'id' and class options that are accepted by each control block to be associated to a certain 'id'.

4.4 The Loader

To build the interface on your basic static HTML page all you need to do is to load the 'loader.js' script from the server into your page. When this file is executed in the client, it loads the other required JavaScript modules, guaranteeing the correct load order, as one module can depend on others.

If no ' ' with the 'id' of a particular module is not found on the page then that module will be disabled.

4.5 Requirements

- SeisComP(reg) 3 provides useful functions for configuration, geometry, travel time computation. If you use the `update-metadata.sh` script, you will need arclink_fetch, either included in the SeisComP distribution, or standalone [http://www.seiscomp3.org/wiki/doc/applications/arclink_fetch].

- Seiscomp Python library ($SEISCOMP_ROOT/lib/python/seiscomp), including a recent version of *manager.py*

  (SeisComP 3 release >= 2013.200; there is a temporary version with this release in the *tools* directory, which you can use to replace your installed version in $SEISCOMP_ROOT/lib/python/seiscomp/arclink).

- JavaScript libraries: jquery-base, jquery-ui [https://jquery.org/]
- OpenLayers. [http://www.openlayers.org/]
- Python, mod_wsgi (if using Apache). Also Python libraries for libxslt and libxml.
- Finally, users' web browsers need to run JavaScript.

4.6 Download

Download the tar file / source from the GEOFON web page at http://geofon.gfz-potsdam.de/software. Or from github at [URL TBD]. [Eventually it may be included in the SeisComP distribution.]

Untar into a suitable directory visible to the web server, such as */var/www/webinterface*:

```
cd /var/www/webinterface
tar xvzf /path/to/tarfile.tgz
```

This location will depend on the location of the root (in the file system) for your web server.

4.7 Installation on Apache

To deploy the WebDC3 web interface on an Apache2 web server using *mod_wsgi*:

0. Unpack the files into the chosen directory. (See "download" above.) In these instructions we assume this directory is */var/www/webinterface*.
1. Enable *mod_wsgi*. For openSUSE, add 'wsgi' to the list of modules in the APACHE_MODULES variable in */etc/sysconfig/apache2*:

   ```
   APACHE_MODULES+=" python wsgi"
   ```

   and restart Apache. You should now see the following line in your configuration (in */etc/apache2/sysconfig.d/loadmodule.conf* for openSUSE):

   ```
   LoadModule wsgi_module   /usr/lib64/apache2/mod_wsgi.so
   ```

   There may be a message like this in Apache's *error_log* file, showing that *mod_wsgi* was loaded:

   ```
   [Tue Jul 16 14:24:32 2013] [notice] Apache/2.2.17 (Linux/SUSE)
   PHP/5.3.5 mod_python/3.3.1 Python/2.7 mod_wsgi/3.3 configured
    -- resuming normal operations
   ```

   Or look at the output from `a2enmod -l` - you should see wsgi listed.

2. Add the following lines to a new file, *conf.d/webinterface.conf*, or in *default-server.conf*, or in the configuration for your virtual host:

   ```
   WSGIScriptAlias /webinterface/wsgi /var/www/webinterface/wsgi/webdc2.wsgi
      <Directory /var/www/webinterface/wsgi/>
          Order allow,deny
   ```

```
        Allow from all
    </Directory>
```

Change */var/www/webinterface* to suit your own web server's needs. You may also need to add a section like:

```
<Directory /var/www/webinterface/>
    Order allow,deny
    Allow from all
</Directory>
```

3. Set environment for Apache: Apache needs the "SeisComP" environment variables set when it starts. The seiscomp3 `bin` and `man` directories aren't needed. For OpenSUSE, add the following lines, which are provided by *seiscomp print env*, to `/etc/sysconfig/apache2`:

   ```
   SEISCOMP_ROOT=/home/sysop/seiscomp3
   LD_LIBRARY_PATH=/home/sysop/seiscomp3/lib
   PYTHONPATH=/home/sysop/seiscomp3/lib/python
   ```

   (Omit "export" and variable references, those will not work.)

   For Debian and Ubuntu/Mint add the following lines to the `envvars` file:

   ```
   # Make SeisComP3 available for webinterface:
   export SEISCOMP_ROOT=/home/sysop/seiscomp3/
   export LD_LIBRARY_PATH=/home/sysop/seiscomp3/lib:$LD_LIBRARY_PATH
   export PYTHONPATH=/home/sysop/seiscomp3/lib/python:$PYTHONPATH
   ```

4. Fix the path which is added in `wsgi/webdc2.wsgi`:

   ```
   sys.path.insert(0, '/var/www/webinterface/wsgi/')
   ```

5. Copy webinterface.cfg to e.g. $SEISCOMP_ROOT/etc, or make a symbolic link from there to the webinterface directory:

   ```
   cd $SEISCOMP_ROOT/etc
   ln -s /var/www/webinterface/wsgi/webinterface.cfg webinterface.cfg
   ```

6. Edit webinterface.cfg. This is discussed under "Configuration Options" below.

7. (TBD: Copy one of the top-level example pages to index.html and customise the site as you wish. October 2013: Just skip this step!)

8. Start/restart the web server e.g. as root:

   ```
   # /etc/init.d/apache2 configtest
   # /etc/init.d/apache2 restart
   ```

9. Get initial metadata in the *data* directory. For instance you can run the `update-metadata.sh` script in that directory.

10. Visit <http://localhost/webinterface>. You should see the front page.

11. Arrange for regular updates of the metadata in the *data* directory. Something like the following lines will be needed in your crontab:

    ```
    # Daily metadata update for webinterface:
    52 03 * * * /srv/www/webdc/webinterface/data/update-metadata.sh
    ```

4.7.1 Installation problems

If you see the basic web interface page, but none of the controls load, you may not have the underlying services running correctly. Look in your web server log files (e.g. for Apache: *access_log* and *error_log*) for clues.

If you visit http://localhost/webinterface/wsgi/loader (or similar) on your machine you should see the definitions that the JavaScript needs to get started:

```
var eidaJSSource='/webinterface/js';
var eidaCSSSource='/webinterface/css';
var eidaServiceRoot='/webinterface/wsgi';
var eidaDebug=false;
$(document).ready(function() { $.getScript(eidaJSSource + '/loader.js') });
```

If these definitions are not found, then you won't have any controls. If they *do* show up, check that the URL paths look correct.

You should also be able to visit the "web service" URLs in your browser e.g. going to:

```
http://localhost/webinterface/wsgi/event/catalogs
```

should show you something like this:

```
{"geofon": {"description": "GFZ (eqinfo)", "hasDepth": true, "hasDate":
true, "hasRectangle": true, ...
```

4.7.2 Configuration options

Configuration follows the SeisComP 3 pattern. Configuration is read from files using a 'dotted' notation e.g.:

```
js.wms.server = "http://myserver.org/wms/vmap0"
```

See the SeisComP documentation [http://www.seiscomp.org/] for details. Configuration variables beginning with "js" are loaded by the JavaScript loader and made available to scripts in the client's web browser. Other variables are only available to the Python-based back end modules.

The following files are sought, and if present, their configuration information is loaded, in the order shown:

1. $SC3ROOT/etc/defaults/global.cfg
2. $SC3ROOT/etc/defaults/webinterface.cfg
3. $SC3ROOT/etc/global.cfg
4. $SC3ROOT/etc/webinterface.cfg
5. $HOME/.seiscomp3/global.cfg
6. $HOME/.seiscomp3/webinterface.cfg

Remember that $HOME is for the *user running webinterface*, which might be the same user as runs your web server. It may be helpful to make a symbolic link from one of these locations to a file in the webinterface directory e.g.:

```
cd /var/www/.seiscomp3 ; ln -s /path/to/webinterface/wsgi/webinterface.cfg .
```

At a minimum, you will need to set *arclink.address* to point to your Arclink server, and EMAIL_ADDR etc. to something suitable for your site. Other options should be suitable for getting started. For full details of all configuration options, see full-config-options.

General options

- Mail server details. WebDC3 sends e-mail to the address given in the Arclink request confirming that the request has been submitted. **FIXME**
- Temporary files. WebDC3 creates files in Python's default temporary directory. This is typically /tmp. This location cannot yet be overridden in webinterface, but you may be able to change it by setting TMPDIR in WebDC3's environment.

Metadata options

- [list of sensor types VBB, BB, OBS etc] to be displayed in the Stations/Streams tool.
- Arclink nodes configuration file: this is an XML file [or a URL?]. This option enables you to give a list of Arclink servers which can be checked for status of requests. Generally this list should be those servers which are included in the routing table provided by your Arclink server. For an EIDA node, this should be the EIDA master table.

Events options

- Event Search Control options:

```
js.events.magnitudes.minimum = 3.0
js.events.depth.minimum = 0
js.events.depth.maximum = 1000
js.events.coordinates.north = 90
js.events.coordinates.south = -90
js.events.coordinates.west = -180
js.events.coordinates.east = 180
```

- Event service configuration options:

```
event.[list of services]
event.names.lookupIfEmpty = True
event.names.lookupIfGiven = False
```

4.8 Customisation

You may safely modify the following to suit your web site needs:

- webinterface.cfg - this was described above. (Location: where SeisComP looks for configuration files.)
- index.html template document. The template must do the following:
  - Make sure jquery gets loaded e.g.:

```
<script src="tools/jquery/jquery-1.9.1.js"></script>
<link rel="stylesheet" href="css/smoothness/jquery-ui.css" />
<link rel="stylesheet" href="css/smoothness/jquery.ui.theme.css" />
<script src="tools/jquery/jquery-ui.js"></script>
<script src="tools/jquery/jquery.cookie.js"></script>
```

  - Make sure OpenLayers gets loaded:

```
<script src="tools/openlayers/OpenLayers.js"></script>
```

  - Load the JavaScript loader:

```
<script src="loadme.js" type="text/javascript"></script>
```

The template should contain " " elements for the JavaScript controls. They should be left empty in the template because their content will be filled by the controls running in the client's brower. The following controls are available:

```
<div id="wi-Console" class="consoleframe"></div>
<div id="wi-StatusListControl" class="frame"></div>
<div id="wi-StatusQueryControl" class="frame"></div>
<div id="wi-StatusFullControl" class="statusframe"></div>
<div id="wi-EventSearchControl" class="frame"></div>
<div id="wi-StationSearchControl" class="frame"></div>
<div id="wi-SubmitControl" class="frame"></div>
<div id="wi-MappingControl" class="frame"></div>
<div id="wi-RequestManagerControl" class="frame"></div>
```

- css/sample.css - Cascading Style Sheet file.

4.9 Maintenance

There may be some temporary files to clean up from time to time. These should be in Python's default temporary directory e.g. `/tmp`.

Metadata may need updating after changes in Arclink inventory - you can safely run the `update-metadata.sh` script at any time to do that. Webinterface creates a processed version of the Arclink XML, but this will be automatically updated each time webinterface notices a new inventory XML file.

CHAPTER

FIVE

DEVELOPER NOTES

5.1 Principles

We use JavaScript-based Dynamic HTML, together with the Python Web Server Gateway Interface (WSGI; **PEP 333**) for thin web services. Our approach is based on "dynamic HTML", in which JavaScript is used to modify objects in the Document Object Model (DOM) of the page displayed in the browser.

There is a modular decomposition into functions related to presentation, events, services, maps, configuration, and data requests from the Arclink server.

- Coding

  We try to comply with **PEP 8** *Style Guide for Python Code* and **PEP 257** *Docstring Conventions*, and use the Python unittest unit testing framework where convenient.

  For JavaScript... anything goes? There is a helper class to control access to the Python modules.

- Documentation

  This documentation is written in a simple mark-up format called reStructuredText <http://docutils.sourceforge.net/rst.html> (reST). The final documentation is generated using Sphinx. Our philosophy follows the SeisComP documentation, described at http://www.seiscomp3.org/doc/seattle/2013.149/base/contributing-docs.html Look in the *descriptions* subdirectory for configuration options etc. relating to particular modules e.g. wsgi/descriptions .

5.2 Interfaces and name spaces

The communication between JavaScript and Python uses HTTP over a web services-like interface. The Python-based web services can run on any port on localhost (not on a different server, due to browser/JavaScript restrictions against cross-site scripting (XSS) attacks.)

The URL specification for services is divided into major subgroups. They are:

```
<wsgi root>/         ## Interface information service / Page generation
<wsgi root>/event    ## Event-related stuff
<wsgi root>/metadata ## Metadata-related stuff
<wsgi root>/request  ## Request submission and status stuff
```

The real services offered by the server are accommodated into one of these subgroups.

Top level specification, major groups:

```
<wsgi root>/                            ## Interface information service / Page generation
<wsgi root>/event                       ## Event related stuff
<wsgi root>/metadata                    ## Metadata related stuff
<wsgi root>/request                     ## Request submission and status stuff
```

5.2.1 Application level

```
<wsgi root>/configuration             ## Get configuration
  Parameters: None
  Response: JSON structure

<wsgi root>/loader                    ## JavaScript code to load the interface
  Parameters: None
  Response: JavaScript method
```

5.2.2 Event level

```
<wsgi root>/event/catalogs            ## List of catalogs available
  Parameters: None
  Response: JSON - a list of catalogs and the features they offer

<wsgi root>/event/<catalog><?parameters> ## Events query for preparing request
  Parameters: start={datetimestring}
              end={datetimestring}
              minlat={float}
              maxlat={float}
              minlon={float}
              maxlon={float}
              minmag={float}
              maxmag={float}
              mindepth={float}
              maxdepth={float}
              limit={int}
              format={string}
  Response: JSON, CSV, raw, or text lists of events

<wsgi root>/event/parse<?parameters>    ## Parse a user-supplied catalog.
  Parameters: informat={string}         ## The INPUT format. So far CSV, later can be QML, GeoJSON
              columns={string}          ## Comma-separated list of columns
                                        ## in the supplied CSV file, from:
                          ('latitude', 'longitude', 'depth', 'time', 'ignore')
              input=<user supplied catalog sent as POST>
              format={string}           ## The OUTPUT format.
  Response: JSON, CSV - a list of events.
```

5.2.3 Metadata level

```
<wsgi root>/metadata/networktypes     ## List of network types
  Parameters: None
  Response: JSON - a list of network types that the system knows.
            Every row contains two columns: ID and DESCRIPTION.
```

```
<wsgi root>/metadata/sensortypes      ## List of sensor types
  Parameters: None
  Response: JSON - a list of sensor types that the system knows.
            Every row contains two columns: ID and DESCRIPTION.


<wsgi root>/metadata/networks<?parameters>  ## List of networks for menus
  Parameters: start={int}
              end={int}
              [networktype={string}]
  Response: JSON - list of networks after filtering based on the parameters.
            Every row contains two columns: ID and DESCRIPTION. The ID is a
            combination of network code and start/end year.


<wsgi root>/metadata/stations<?parameters>  ## List of stations for menus
  Parameters:
              start={int}
              end={int}
              networktype={string}
              [network={string}]
  Response: JSON - list of stations after filtering based on the parameters.
            Every row contains two columns: ID and DESCRIPTION. The ID is a
            combination of network code, start/end year of the networks and
            the station code.
            If the optional network parameter is passed, networktype is not
            used and the time to return the information is improved.


<wsgi root>/metadata/streams<?parameters>   ## List of streams for menus
  Parameters:
              start={int}
              end={int}
              networktype={string}
              [network={string}]
              [station={string}]
  Response: JSON - list of streams after filtering based on the parameters.
            Every row contains only one column: DESCRIPTION. The DESCRIPTION
            is only the first two letters of the stream code.
            For example, 'BH'.
            If the optional network and/or station parameters are passed, the
            time to return the information is improved.


<wsgi root>/metadata/query<?parameters>     ## Metadata query for preparing
                                            ## request
  Parameters: start={int}
              end={int}
              [network={string}]
              [networktype={string}]
              [streams={list of streams}] ## two-char codes separated by commas
              [station={string}] ||       ## selection "by Code"
              [minlat={float}             ## selection "by Region"
               maxlat={float}
               minlon={float}
               maxlon={float}] ||
              [minradius={float}          ## selection "by Event"
               maxradius={float}          ## Radius and azimuth are
```

```
               minazimuth={float}         ## relative to event location(s)
               maxazimuth={float}
               events=<event data sent as POST>]
              sensortype={string}
              preferredsps={float}
  Response:   JSON list [ key table, attribute list]
              Keys are (network code, network starting year, station code,
                        station starting DATE)
              Attributes are {'netcode', 'statcode', 'latitude', 'longitude',
                             'restricted', 'netclass', 'archive',
                             'netoperator', 'streams' e.g. ["BH","HH","LN"]}
```

NOTE: Coordinates for region-based constraints are taken from <station>-level attributes in inventory. This is not perfect (they are stream-level attributes), but is simple and adequate.

```
<wsgi root>/metadata/phases          ## List the phases supported by the timewindows method.
  Parameters: None
  Response: JSON - a list of phases that the system knows.
            Every row contains two columns: ID and DESCRIPTION.


<wsgi root>/metadata/export<?parameters>          ## Downloads a CSV file with the selected streams
  Parameters: streams={data}         # JSON [list of [net, sta, chan, loc]]
  Response: CSV - a list of streams that are currently selected in the stations list.


<wsgi root>/metadata/timewindows<?parameters>     ## Prepare time window for each (event, stream)
  Parameters: streams={data}         # JSON [list of [net, sta, chan, loc]]
              [ start={time}
                end={time} ] ||
              [ events={data}       # JSON [list of [lan, lon, depth, time]]
                startphase={string}
                startoffset={int}
                endphase={string}
                endoffset={int} ]

  Response: JSON object, suitable to be passed as "timewindows" to /request/submit
```

5.2.4 Request level

(Magic free, all parameters are mandatory unless it is explicitly stated that they are optional.)

```
<wsgi root>/request/types                   ## List of possible request types
  Parameters: (none)
  Response: JSON [list of [requesttype, description]]

<wsgi root>/request/nodes                   ## List of nodes from network XML
  Parameters: (none)
  Response: JSON [list of [DCID, name]]

<wsgi root>/request/submit<?parameters>     ## Submit one request
  Parameters: user={string}
              requesttype={string}
              compressed=<boolean>
              responsedictionary=<boolean>
              timewindows=<data>            ## JSON list of time windows
              eventinfo=<data>       ## Event info, e.g. for SEED headers.
```

```
                                        ## JSON dict: [latitude, longitude, depth, description, ...]
                                        ## (Not currently used.)
              [server={DCID}]

  Response: JSON dict {"uuid": "<uuid>",
                       "success": [list of successfully routed requests],
                       "failure": [list of requests that could not be routed]}
```

Time windows structure example (might use Arclink time format instead of ISO):

```
[
   [
       "2013-07-01T12:00:00.0000Z",
       "2013-07-01T12:10:00.0000Z",
       "RO",
       "MLR",
       "BHZ",
       "",
       1024                                # estimated size
   ],
   [
       "2023-07-01T12:00:00.0000Z",
       "2023-07-01T12:10:00.0000Z",
       "RO",
       "MLR",
       "BHZ",
       "",
       1024                                # estimated size
   ]
]

<wsgi root>/request/download?{parameters}    ## Get the data!
  Parameters: request={int}    ## Arclink Request id number
              user={string}    ## Valid e-mail address
              server={string}  ## Arclink network DCID
              [volume={string}]## Optional volume to download
  Response: (data: seed, mseed, dseed, inv-xml, routing-xml, ...?)

<wsgi root>/request/status<?parameters>     ## Check status of one user request
                                            ## at a server.
  Parameters: user={string}
              server={string}
             [ request={int} ] ## OPTIONAL, if omitted: all request ids for
                                  that user and server should be returned
  Response: JSON list of objects generated from Arclink status XML file for each request

<wsgi root>/request/resubmit<?parameters>
  Parameters: user={string}
              uuid={request UUID}
              mode={reroute|retry|resend}
              idlist                  ## JSON [list of [server, request]]
              [server=DCID]           ## if omitted, use DEFAULT_SERVER

  Modes: reroute: try to send NODATA/RETRY lines to next server;
         retry:   try to send NODATA lines to next server,
                  retry RETRY lines on the same server;
         resend:  resend the whole request under a new UUID.
```

```
  Response: JSON dict, same as /submit

<wsgi root>/request/purge{?parameters}      ## Delete one user request
                                            ## at a given server.
  Parameters: user={string}                 ## User is an e-mail address
              server={string}               ## Arclink network DCID
              request={int}
  Response: true (or HTTP error)
```

As for FDSN web services, `{datetimestring}` is *always* an ISO-style date-time string, e.g. `2010-01-01T12:34:56`. Note that there is a 'T' between date and time and there is no time zone indication. All times are UTC.

5.3 Modules

1. *Presentation module*
2. *Events module*
3. *Station metadata module*
4. *Maps module*
5. *Data requests module*
6. *Configuration*

5.3.1 Presentation module

There is a 'debug' option for the JavaScript.

Styles for objects within the control divs is provided in *wimodule.css*. These are prefixed with "wi-". Generally they do not alter colours or fonts - those should be controlled by a "theme"-specific style sheet included by the top-level page (e.g. index.html includes css/basic.css).

Some things `are` set in the JavaScript e.g. monospace font for pull-down menus.

In particular lots of alignment decisions - padding, margins, text alignment are made in the JavaScript - but they should not require modification.

5.3.2 Events module

Each target event service requires a URL at which we can obtain CSV output.

One difficult choice concerns interpretation of end dates and the time (instant) that they refer to.

Several existing catalogs understand an end date as the last day on which an event should be returned - GEOFON's *datemax* and EMSC's *end_date* parameters are like this. Others specify a time, e.g. ComCat requires milliseconds since 1970. A box giving start and end dates on a web interface needs to convert its end date to the last acceptable time, or the *end* of the end date. For events in June 2013, users should enter 2013-06-30 in an "ending date" box, but this means that the end parameter value required is `2013-07-01T00:00:00`. Following the FDSN web services specification, this date-time may be abbreviated to *end=2013-07-01* i.e. the *start of the next day*.

We were faced with two unpalatable choices:

- Allow *end=YYYY-MM-01* to mean the end of the first day of a month. Then it could be passed through to those target services which cut off at the end of the given date. However the Python web service we built to wrap multiple event services would be flawed in that it did not itself offer FDSN-style date support. Meanwhile interactions with new target services using the FDSN convention would have to compute *end=YYYY-MM-02T00:00:00*.
- Bite the bullet now, and have the Python web service present an FDSN-style interface. Then the JavaScript in the client must prepare an "end={value}" string for sending to the Python, and this must be converted to the older convention for older target services. These two conversions, one in JavaScript, one in Python, increase the possibility of coding errors.

We chose the second option, so that

1. we can support *times* within days in future (e.g. a search for events between 00:00 and 06:00 on 1 April 2013), and
2. we stop perpetuating the same problem of being unclear about what an incompletely-specified time like 2013-04-01 means.

The event service handlers for GEOFON and EMSC now convert a request like:

```
/event/geofon?end=2013-04-01
```

into requests for

<http://geofon.gfz-potsdam.de/eqinfo/list.php?datemax=2013-03-31>

<http://www.emsc-csem.org/Earthquake/?filter=yes&end_date=2013-03-31>

Implementing extensions

To build a new event service:

1. Add it to the list of configured event services, so that the front end displays it. For now, selecting this will do nothing, at best, and probably crash your browser. :-)

   For now, add it in _EventsServicesCatalog in the Python (event.py)

2. Add some test cases in test/testEvent.py.
3. In the Python (event.py) file: subclass EventService. You need to provide a handler() method. This function is expected to:

   (a) Builds a query from the parameters it receives.

   (b) Query the target service

   (c) Process the response to produce a JSON object representing a list of events.

   (d) Return this object to the caller.

   The EventService class provides several methods to help you do this.

   - result_page()
   - format_response()
   - error_page()
   - the process_parameters function.

   A simple event service class definition might be:

```
class ESPhlogiston(EventService):
  def __init__(self, name):
     # Accept defaults, or they can be overridden here:
     self.csv_dialect = ...
     self.column_map = ...
     self.filter_table = ...

  def handler(self, environ, parameters):
     """Get events from http://quakes.phlogiston.org/ as CSV."""

     # set paramMap to handle the FDSN-style parameters from the QUERY_STRING
     pairs, bad_list, hold_dict = process_parameters(paramMap, parameters)

     # Build URL ready for submission and make the request
     try:
       allrows, url = self.send_request(pairs)
     except urllib2.URLError:
       self.raise_client_400(environ, 'No answer')

     fmt = hold_dict.get('format', 'text')
     content = self.format_response(allrows, numrows, limit, fmt)
     return self.result_page(environ, start_response, '200 OK', 'text/plain', content)
```

The 'parameters' dictionary contains values for zero or more of the following arguments:

Parameter name	Allowable values	Remarks
start	[0-9TZ:-.]*	
end	[0-9TZ:-.]*	
minlat	float [+-][0-9.]	
maxlat	float	
minlon	float	
maxlon	float	
lat	float	
lon	float	
minradius	float_pos	
maxradius	float_pos	
mindepth	float	Negative depth is okay
maxdepth	float_pos	
minmag	float	Negative mag is okay
maxmag	float_pos	
magnitudetype	string???	Can we do wild cards? What do MT solutions have?
preferredonly		Ignored for eqinfo
eventid		Sure, might be handy
includeallmagnitudes	bool	NOT supported
includearrivals	bool	NOT supported
limit		
offset	•	Not in eqinfo
orderby	•	could be
contributor	•	Not in eqinfo
catalog	•	ignored
updatedafter	•	Ignored for now (EMSC?)

The values of these parameters (default, type, units etc.) are as set out in the FDSN standard. In particular date-time strings with no time refer to the start of the day e.g. "end=2000-04-01" implies "2000-04-01T00:00:00.0", the *start* of 1 April, not the end of this day.

[Extension: lat and lon may be vectors (values separated by commas). If both have the same number of items, then each lat-lon pair is checked in turn in searching for matching events. It is an error if the lists are of different length, or if one of lat and lon is not present when the other is.] [The long (unabbreviated) parameter names in Table 1 are *not* supported.]

These are passed to getEvents() during an event services request, as arguments to the URL. The similarity to the FDSN 'event' web service *is* intentional. [1] Some parameters in the FDSN 'event' web service are not relevant to the web interface at present, or are not implemented in the GEOFON eqinfo service, but it should be okay to include them in queries.

The columns of the CSV list of events must be in the following order:

[1] See Table 1 of "FDSN Web Service Specifications", Version 1.0, 2013/04/24, accessed 2013-10-10 from http://www.fdsn.org/webservices/FDSN-WS-Specifications-1.0.pdf . We do not claim to support the entire FDSN-defined service interface. A major difference between this web service and FDSN's is that FDSN web services are expected to return parametric data for events as QuakeML - any text/CSV output is an undocumented extension of the FDSN interface. Furthermore, our services, at this stage, are not available to the general public, or even necessarily hosts beyond *localhost*.

Note: Put this table elsewhere.

Table: Existing/Proposed/to be implemented event services:

Service Name	Status	Description
geofon	Done	GFZ eqinfo service
comcat	Done	USGS, replaces NEIC
emsc	Done	EMSC
parse	Done	Event time, lat/long by hand on web page
iris	TODO	Text-based
file-txt	TODO	Text file upload
file-qml	TODO	QuakeML file upload
fdsn-qml	TODO	Generic QuakeML-based service
sc3-txt		

- **geofon:** Our GFZ eqinfo service (text services need tuning per supplier).
- **EMSC:** Old pre-FDSN web service at <http://www.emsc-csem.org/> -have CSV and JSON and (pre?)QuakeML Base: <http://www.emsc-csem.org/Earthquake/?filter=yes&export=csv>
- **NEIC: reserved for old service at <http://neic.usgs.gov/>** Base: <http://neic.usgs.gov/cgi-bin/epic/epic.cgi?> + *SEARCHMETHOD=1&FILEFORMAT=6&SEARCHRANGE=HH {params} &SUBMIT=Submit+Search*
- **sc3fdsnws-txt:** Talk to a SC3 implementation of FDSN web services, using fmt=txt option.
- **fdsnws-qml:**

  Talk to a generic implementor of FDSN web services, using QuakeML.
  {baseUrl}/fdsnws/event/1/query? {params} &format=&nodata=

The following table shows how some non-standard services are implemented:

Table: Event service mappings

FDSN Standard	GFZ eqinfo	EMSC	NEIC (old)[2]
start	start	start_date	SYEAR,SMONTH,SDAY
end	end	end_date[*]	EYEAR,EMONTH,EDAY
minlat	latmin	min_lat	-unavailable
maxlat	latmax	max_lat	
minlon	lonmin	min_long	SLON1 ?
maxlon	lonmax	max_long	SLON2 ?
lat	-unavailable	-unavailable	CLAT
lon	-unavailable	-unavailable	CLON
minradius	-unavailable	-unavailable	
maxradius	-unavailable	-unavailable	
mindepth	-drop[1]	min_depth	NDEP1=0
maxdepth	-drop	max_depth	NDEP2=depth
minmag	magmin	min_mag	LMAG ?
maxmag	-unavailable[3]	max_mag	UMAG=9.9 ?
•		min_intens	
•		max_intens	
•		region	
magnitudetype		-unavailable	
preferredonly		-unavailable	
eventid	“”	-unavailable	
includeallmagnitudes	-unavailable	-unavailable	
includearrivals	-unavailable	-unavailable	
limit	nmax	“”	
offset	-unavailable	“”	
orderby	-unavailable	-unavailable	
contributor	“”		
catalog	“”		
updatedafter	-unavailable	“” [1]	

- “-unavailable” : submission with this parameter would be ignored, result in bad/misleading results, is an error, don’t submit.
- “” : harmless, pass this parameter on to target, but it won’t be processed by it.

Note 1: ‘updatedafter’ is an attribute present in EMSC output, but is not constrainable in query parameters. ‘depth’ is present in eqinfo output, but is not constrainable.

Note 2: Looks like NEIC had no geographical constraints, hence filterEventsFromNEIC did it in the old js/query.js.

Note 3: We added magmax to the eqinfo service to implement this (July 2013).

3. Implement the functionality

You may need to rename arguments passed to, and reorder outputs etc. received from your target service. This wrapper function achieves that. Regarding output, see below.

If you encounter an error while querying your target service, simply return an empty string. The getEvent function calling yours will see this response and generate a “204 No Content” response, and the web interface will report to the user that no events were available for the selected event catalogue and parameters.

4. Add an instance of the new class in getEvents() in *events.py*.

#. Add one or more test functions for your function in the TestEventServices class (*test/testEvents.py*), run the unit tests and start the service stand-alone:

```
cd test
python testEvents.py
python testMetadata.py
python manage.py
```

You can now try the service, by visiting <http://localhost:{port}/event/{service}?{params}>

1. Add a new option to generate a different `<div>` e.g. a file upload box, or a picker for single event!
2. Restart WSGI on the server, refresh or close your browser to reload JavaScript.
3. Check that the new module works as expected. Debugging info goes to Apache's logging (typically */var/log/apache2/error_log*) and SeisComP's logging (which may be in *~/seiscomp3/var/log*, depending on your configuration.)

Event service output

The event service JSON output must have the structure of a table with one row per event. The columns in each row are:

Position	Quantity	Type	Remarks
0	Event Time	datetime	Rounded to seconds!
1	Magnitude	float/str	"–" if not in input
2	Latitude	float	
3	Longitude	float	
4	Depth	float/str	"–" if not in input
5	Event ID	string	Used by JavaScript
6	Region	string	Can be filled if missing

Rounding event times to the closest second might have consequences if users expected waveforms to be very carefully aligned.

5.3.3 Station metadata module

The information related to the inventory is first retrieved and updated from an Arclink server by means of a script (*data/update-metadata.sh*) installed in a crontab. The update interval can be configured according to the needs of the operator. As this information is does not change frequently over time, an update interval of 24 hours is the suggested value.

This information is saved to a file on the server (*data/Arclink-inventory.xml*) and will be read from this file if necessary. The parsing of the file and the creation of the internal representation can take up from 5 to 9 seconds, depending on the hardware. To improve performance, once the information is stored in memory, a dump of all these variables are saved in a temporary file. If other threads of the server are started, the timestamp of the original information and the memory dump is checked and the newer is loaded. In this way, the system does not need to establish a connection with the Arclink server while consulting the metadata, making operations much faster than in the previous version of the system.

The internal representation of the metadata consists of four lists representing networks, stations, sensor locations and streams. All the lists contain tuples and every tuple represents one instance of the related information (e.g. one network). The structure of these tuples is described below.

Network:

Position	Variable	Type	Remarks
0	Code	string	
1	First station	int	Pointer to the first station of the network. If it is a virtual network, this should be None.
2	Last station	int	Pointer to the last station of the network (exclusive; to be used with the function range). If it is a virtual network, this should be None.
3	Stations	list	Station pointers in case of a virtual network.
4	Start year	int	Start year of operation.
5	End year	int	End year of operation.
6	Description	string	
7	Restricted	boolean	True if the network has restricted data.
8	Class	char	'p' for permanent and 't' for temporary.
9	Archive	string	Archiving node, 'GFZ', 'RESIF', 'INGV', etc.
10	Institutions	string	Network operators.

Station:

Position	Variable	Type	Remarks
0	Network	int	Pointer to the containing network.
1	First sensor	int	Pointer to the first sensor of the station.
2	Last sensor	int	Pointer to the last sensor of the station. (exclusive; can be used with *range*).
3	Reserved	NoneType	
4	Code	string	Station code.
5	Latitude	float	
6	Longitude	float	
7	Description	string	
8	Start date	datetime	Start date and time of operation.
9	End date	datetime	End date and time of operation.
10	Elevation	float	

Sensor Location:

Position	Variable	Type	Remarks
0	Station	int	Pointer to the belonging station.
1	First stream	int	Pointer to the first stream of the sensor.
2	Last stream	int	Pointer to the last stream of the sensor. (exclusive; can be used with *range*).
3	Reserved	None-Type	
4	Code	string	Sensor code.

Stream:

Position	Variable	Type	Remarks
0	Sensor	int	Pointer to the belonging sensor.
1	Code	string	Stream code.
2	Sensor type	string	
3	Sample denom.	float	
4	Sample numer.	float	
5	Datalogger	string	
6	Start date	datetime	Start date and time of operation.
7	End date	datetime	End date and time of operation.

5.3.4 Maps module

Uses OpenLayers.

The icons supplied for station and event markers are 13x13 PNG images, with an alpha channel. They were produced using Inkscape.

5.3.5 Data requests module

This part communicates with the Arclink server. It can break large requests into chunks and handle splitting requests between servers.

FIXME: Andres, does “reroute” work down the routing table in priority order, or something else?

5.4 Configuration

webinterface.cfg is processed using SeisComP configuration code. It is read in by ...XXX. Configuration values can be obtained in the Python code by ... XXXX.

CHAPTER

SIX

CHANGE HISTORY

6.1 Initial revision

Compared to the old web interface, developed until 2012, there are many internal and external differences.

- Site customisation: There's no more html_chunks.py! Instead just create index.html containing the hooks you need. See *Customisation*.
- Modular support for multiple event catalogs.
  - The old NEIC service has been removed, and replaced by USGS's Comcat service.
  - EMSC's service has been added.
  - You can upload events from a custom CSV catalog.
- Stations/metadata:
  - Webinterface caches inventory data from its base Arclink server. This saves many Arclink requests and gives much better performance. The *update-metadata.sh* script is provided as a tool for refreshing locally downloaded data.
  - There is better display of what streams are available for a particular set of stations.
  - Selection based on sample rate closest to a specific value is possible.
- Request handling:
  - Request status is displayed for all data centres which were involved in handling a request.
  - Routing/retrying/refreshing.
- This documentation exists.

6.2 v0.3 (2013-10-11)

- js - event services URL can have a limit. Quick M>=6 feature. Extra help comments. Different event symbols.
- For events and stations, 'Search' -> 'Search/Append'
- For events and stations, items which are unselected remain visible on the map.
- Add 'restricted' column on station table.
- Request metadata/streams by POST not GET.
- Warning on too many unpurged requests at the data centre.

- Swap “Time Window selection” and “Request information” on submit tool.
- Added ‘type=”text/javascript”’ to <script> elements in HTML documents.
- Documentation: Many small improvements and updates.

CHAPTER

SEVEN

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

APPENDIX

A

SELF-STUDY TUTORIAL

Peter L. Evans, GEOFON team pevans@gfz-potsdam.de

A.1 Introduction

The GEOFON WebDC3 web interface is a new and powerful web-based tool to explore earthquake event catalogs, browse seismic stations, and extract seismic waveforms and station metadata from the EIDA archive system. It extends the old WebDC service in several interesting ways.

This document is intended to be a self-study guide to performing a few common tasks with the web interface tool, and demonstrating the new features. It is task-driven: you will be asked to find a few sets of data, and guided through the way where necessary. The whole set of tasks shouldn't take more than half an hour (it takes me a few minutes to click here and there, but then I wrote this document, and helped develop the software.)

Each task below is focussed on extracting some specific information from the system. There is a question for each one. Answers are at the back. No peeking! If you encounter problems along the way, it may be because the interface is unclear, the documentation in the User Guide is incomplete, or there are bugs in the software. *Please* let us know about your experience. You can send e-mail to geofon@gfz-potsdam.de, or contact me at the address above.

If you're stuck, note the on-line help available as pop-ups (or whatever). Also you can click on the link in the top right corner of the page.

To start, open your web browser on to the web interface start page, either at [http://eida.gfz-potsdam.de/webdc3] or at your local site.

A.2 Event browsing

Q: *In July 2013, how many big earthquakes were there, worldwide?*

Click on the "Explore events" tab at the top of the page. You will see a box titled "Events Controls". Use this to make a selection of events in the GFZ event catalog. Press "Search" when you are ready. You should see a list of events and they are displayed on the map.

1. How many had magnitude >=6?
2. How many had magnitude >=5.5?
3. (Harder) How many of these are also in the EMSC and USGS seismic catalogs?

Clear your selection of events (click "Delete Events"). Now we'll ask a more specific question:

4. How many events are recorded near Tonga (Nukualofa, 21 degrees S latitude, 175 degrees W longitude, within say 5 degrees) with M>4? Of these, how many have depth between 100 and 400 km?

A.3 Station browsing

Q: *How many stations are there in the GEOFON seismic network?*

Click on the "Explore stations" tab to expand the "Stations Controls" box.

1. How many stations were in the GEOFON network (network code "GE") in 2013? According to inventory, how many of these had BH stream data:
   (a) Based on channel codes? [Use "by Code".]
   (b) Based on sample rate close to 20 sps? Is there a difference? (Hint: see the help page.)
2. How many of these stations have STS-2 instruments? *This one can't be done with our first version.*
3. Press the "Reset" button in the Station Controls. For the network GE station APE (Apirathos, Naxos, Greece), how many channels are available altogether?
4. How many stations were in the GE network in 2003?
5. [What about something EIDA-wide too? For this you need "All shared networks".]

A.4 Requesting waveform data

Q: *What waveforms do you have for my event?*

Reload the page. Request mini-SEED waveform data for all Mediterranean broadband stations (within 4 degrees) which recorded the M5.0 event in Central Italy on 2013-07-21. Under "Explore Stations", use the "by Regions" button to filter stations by region. Restrict your selection to just BH channels.

Use the "Submit request" tab. Request just the vertical component (BHZ) using "Filter" on the station list. Use "Relative mode" on the "Submit request" tab to set time windows from 1 minute before the expected P wave arrival to 5 minutes after the expected S wave arrival for each station.

Request full SEED waveform data. Click "Review request". Once your request is sent, use the "Download data" tab to see how your request is progressing.

- how many streams did you obtain?
- how many time windows, z, are there in your request?
- how many time windows, y, are in your request (use the "Review" button)?
- how many time windows, x, returned data (use the "Status" tab)?
- what is the size of the file you downloaded?

Note that $x <= y <= z$ because:

1. A time window with a P arrival can't be computed for all stations.
2. We have no data from some stations at the times requested.

A.5 Requesting station metadata

Q: *I need to set up my new SeisComP system. How do I get the metadata I need?*

One way to do this is via Arclink inventory XML. On the Station Controls, select for years 2011 to 2013. Pick the GEOFON network (code GE) from the list under "Code". Under "Submit request", pick "Metadata (Inventory XML)" (and "Absolute Mode").

Another way is via dataless SEED. Which is smaller?

A.6 Request status and cleaning up

Q: *What's the status of my request?* You can also see what requests are pending, i.e. haven't been completed, and are available for downloading. Go to the `Download data` tab.

Click on a line starting "Package..." to see its status. Use the "Refresh" button, and for a big request, you may notice the number of lines with "Status: PROCESSING" increases, while that with "Status: UNSET" decreases. When everything is done, you will see "Status: OK" and green text "Download Volume". Clicking on this text lets you save the data to your local computer.

A.7 Using catalog upload

Q: *But I have my own event catalog! Can I still use the web interface?*

On 15 February 2013 a meteor exploded over Chelyabinsk, Russia [http://en.wikipedia.org/wiki/Chelyabinsk_meteor]. What waveform data do you have around this time? A quick look in the GFZ catalog shows we have no event associated with this meteor. Create a custom event by choosing "User Supplied" in the Event Controls box. Use depth 0 and time 03:20 UTC.

Now download BHZ data for stations within 90 degrees of 55.0 degrees N, 61 degrees E.

[I need a good simple way to view SEED data.]

A.8 Data at different EIDA nodes

Q: *Isn't there more than one EIDA node?*

Within the EIDA system, waveform data may be stored at only one participating EIDA node, but it is still available from the web interface running at GFZ or other nodes. For example seismic network CH is hosted at ETH in Z?rich, while GE data is here at GFZ. Request BHZ/HHZ waveforms for all stations in [a box from 45 to 55 degrees N, 5 to 15 degrees E - including some German stations.] Note that GEOFON station GE.RUE, XXX and XXX are included - data for these is stored at GFZ Potsdam and BGR Hannover/LMU Munich respectively. As a time window, take the first 15 minutes of April 1, 2013. How many streams are in your request?

Note that your request is broken into volumes and sent to each node. You can see the status of each one using the `Download data` tab.

A.9 Direction-based searches

Find all stations to the north (i.e. azimuth between 330 and 30, distance less than 120 degrees) of any South American event with M>6.0 between January 1 and March 31 of 2013.

You must first select the events, from the `Explore events` tab. Then use the "Explore Stations tab to go to the Stations Controls. Select "by Events" and the desired event distance and azimuth.

A.10 Last words

Finally, clean up your requests, after downloading them. (From `Download data`, put your e-mail address in the Manage Requests box, and click "Get Status", to see all your requests at all EIDA nodes. Now you can delete them all when you are ready.)

Thank you for working through this document. Here are the final questions:

1. How long did it take you to work through these tasks?
2. How can we improve the web interface?
3. Can you give us an example of a request you would often like to make, but can't today?

The answers to these questions are **not** provided below.

A.11 Answers to exercises

Note: The specific event numbers, stream details, file sizes etc. listed here were accurate for the GFZ web interface at http://eida.gfz-potsdam.de/webdc3 in August 2013. They may have changed by the time you work through this document.

A.11.1 Event browsing

1. 13 with M>=6.0; there is 1 with M5.9, and 1 with M5.8
2. 33, plus 5 with M5.4; but setting magmin=5.4 gives 39!
3. For M>=6.0 there are also 13 with all catalogs; but differences can occur when the magnitude is close to the threshold. For M>5.5, EMSC has 40 events, while USGS has 33.
4. 15, and 2 (at 2013-07-24T03:32:33Z and 2013-07-30T03:00:32Z).

A.11.2 Station browsing

1. In 2013: 75 stations; but this may increase during the year. [There might be a difference between these two ways of selection.]
3. 5 x 3 components = 15 channels.
4. In 2003 there were 52 stations.

A.11.3 Requesting waveform data

The event in question has latitude 43.56N, longitude 13.76E. I found 333 stations in inventory, from at least 12 different networks, within 4 degrees. For BH streams, there are 189 stations.

There are hundreds of stations you could use for this. Out of my selection of 189 stations, filtering down to BHZ built a request with $y = 185$ traces, with one time window per station.

(Since much of the data is at INGV, not GFZ, there may sometimes be routing problems in fulfilling your request.)

A.11.4 Requesting station metadata

For GE metadata for 2011-2013, my inventory XML file was 760 kB for 1383 streams in 75 stations. The corresponding dataless SEED file was about 512 kB.

A.11.5 Direction-based searches

Use longitude 85W to 25W, latitude 60S to 15N; there are only 2 events. I found over 130 matching stations, from networks 5E, CN, CX, DK (Greenland), G, GE and others.

INDEX

P