FISSURES-MATLAB INTERFACE

Qin Li, Ken Creager*** and *Zuoli Ning***
* Department of Electrical Engineering
** Department of Earth and Space Sciences
University of Washington
October, 2003

1. INTRODUCTION

FISSURES is an effort to define object-oriented seismic classes so that software developers can use standard objects for seismology. Additionally FISSURES uses a distributed computing technology called CORBA (Common Object Request Broker Architecture) to allow software systems to work across the Internet in a platform independent and computer-language neutral manner. The core of FISSURES framework is written by IDL (the Interface Definition Language). Basically IDL can be compiled to many languages, like C++ and Java. Due to the superb compatibility of Java across different platforms, we use the pre-compiled Java version that is being maintained at the University of South Carolina (http://www.seis.sc.edu/software/Fissures/library.html).

MATLAB® is a scientific/engineering computing language that is very popular among seismologists. MATLAB provides support for Java v1.3 since release 13 (version 6.5). Directly calling functions in FISSURES framework from MATLAB thus becomes possible. FISSURES-MATLAB Interface (FMI) is an effort to build a bridge between FISSURES and MATLAB. It allows MATLAB users to take advantage of FISSURES framework to handle seismic data without knowing the details of FISSURES and java coding, and to easily build their own clients.

2. OVERVIEW

Like FISSURES, FMI is based on object oriented programming (OOP) approaches. The data and the functions operating on the data, which are called methods, are encapsulated in MATLAB objects. Corresponding to the three FISSURES compliant services, the event service, the network service, and the waveform service, provided by IRIS DMC, three MATLAB classes are built in FMI. An overview diagram of FISSURES and FMI is shown in figure 1.

2.1 MatEvent

MatEvent is a MATLAB class which provides functionalities of querying, retrieving, and mapping earthquake information, and contains data structures that store this information. The structure of MatEvent is shown is figure 2.

2.2 MatNetwork

MatNetwork is a MATLAB class which includes data structures for network, station, and channel information. It provides many methods for querying, retrieving, mapping stations/channels, as well as instrument response retrieving and calculating. The structure of MatNetwork is shown is figure 3.

2.3 MatSeismogram

MatSeismogram is a MATLAB class which includes methods to query and retrieve seismogram, and data structures to store seismograms and their attributes. The structure of MatSeismogram is shown in figure 4.

3. SYSTEM REQUIREMENTS

FMI can be run on any system with MATLAB version 6.5 (R13) or high installed. The Java Virtual Machine (JVM) must be enabled. A system with 500MHz CPU or above, and 256 MB memory or above is recommended. FMI has been successfully tested on Mac OS X 10.2 & 10.3, Soaris, Windows (2K, XP), and Linux (Redhat).

4. HOW TO USE FMI

Please look at FMI references section at the end of this document for details. Although it is not required for FMI users to know details of MATLAB OOP approach, reading documentations for MATLAB OOP approach will be very helpful to understand the FMI reference.

There are several base methods for all MATLAB classes, but only three are implemented, `get`, `subsref`, and `display`. In MABLAB objects, all data are hidden from outside of the objects. Method `get` and `subsref` provide two ways to read data from an object. The general usage of `get` method is `get(object, 'property_name')`. The method `subsref` is called implicitly whenever an object is followed by a subscripted reference in an expression. For example, if `myEvent` is a `MatEvent` object, then `myEvent.preferredOrign(i)` will invoke an automatic call to method `subsref`, and it will return the preferred origin for the i^{th} event. So `subsref` offers a convenient way to read data in objects just like accessing structure fields and array elements. Method `display` is called whenever MATLAB displays the contents of an object (e.g., when an expression is entered without terminating with a semicolon).

5. DEMOS AND APPLICATIONS

EventDemo: A simple command-line demo of using MatEvent

NetworkDemo: A simple command-line demo of using MatNetwork

SeisDemo: A simple command-line demo of using MatSeismogram

EventFinder: A GUI (Graphical User Interface) program to query events by magnitude, geographic area, depth, time and catalog type. It is retrieves event information from an IRIS DMC server, and plots event maps.

SeisFinder: An event oriented GUI program to query seismograms maintained at the IRIS DMC by network/station/site/channel and time window. It is also able to retrieve seismograms, their attributes and channel information of the seismograms.

6. OTHER TOOLBOXES USED IN FMI

Three third-part toolboxes are used in FMI.

TauP: TauP is a Java based toolbox to calculate traveltimes, raypaths, and many other parameters related to seismic ray propagation. TauP is written and maintained by H. Philip Crotwell, Thomas J. Owens, and Jeroen Ritsema in the University of South Carolina (http://www.seis.sc.edu/software/TauP/index.html).

M_Map: M_Map is a mapping package for MATLB written and maintained by by Richard Pawlowicz (http://www.eos.ubc.ca:/~rich). The stand m_map toolbox only includes a very coarse coast line data file. The high-resolution GSHHS coastline data files are optional. You can download them from FMI website or from University of Hawaii website (http://www.soest.hawaii.edu/wessel/gshhs/gshhs.html).

MatTaup: MatTaup is a set of MATLAB wrapping programs of TauP toolboxes written by Qin Li at the University of Washington and distributed with FMI.

7. LIMITATIONS AND KNOWN BUGS

Please note that MatNetwork class has not been fully tested and debugged. The instrument response processing and handling have not been implemented. The complete MatNetwork class will be included in the next release.

Although MatSeismogram supports querying and retrieving seismograms from POND, Archive, and BUD data servers, the methods are optimized for POND. Querying and retrieving data from Archive server may be very slow and cause MATLAB lose response. Optimized methods for Archive and BUD servers will be included in the future. Currently we suggest using POND only.

We have noticed that when we query available seismograms, occasionally the time window of the available seismogram returned from DMC is wrong, which will cause a Java exception when retrieving the station location. This has been confirmed a bug in the seismogram server in DMC. We observed this problem occurring at 1% to 2% of total available seismograms. However, if this problem happens, the Java exception will be properly caught and handled without interrupting FMI programs. The seismogram with wrong time window will be threw out.

Sometimes when you try to retrieve a big number of seismograms, say more than 500, at one time, the Java "out of memory" exception may happen. If this happens, try to break your query into several small queries, or increase your computer memory.

8. LICENSE

FMI and applications based on FMI are free to download and use under the terms of Gnu Public License (http://www.gnu.org/copyleft/gpl.html).

9. CONTACTS AND BUG REPORT

For general information, contact Ken Creager (kcc@ess.washington.edu)
For FMI bugs, contact Qin Li (qinli@u.washington.edu)
For bugs of EventFinder and SeisFinder, contact Ronnie Ning (ronnie@ess.washington.edu)

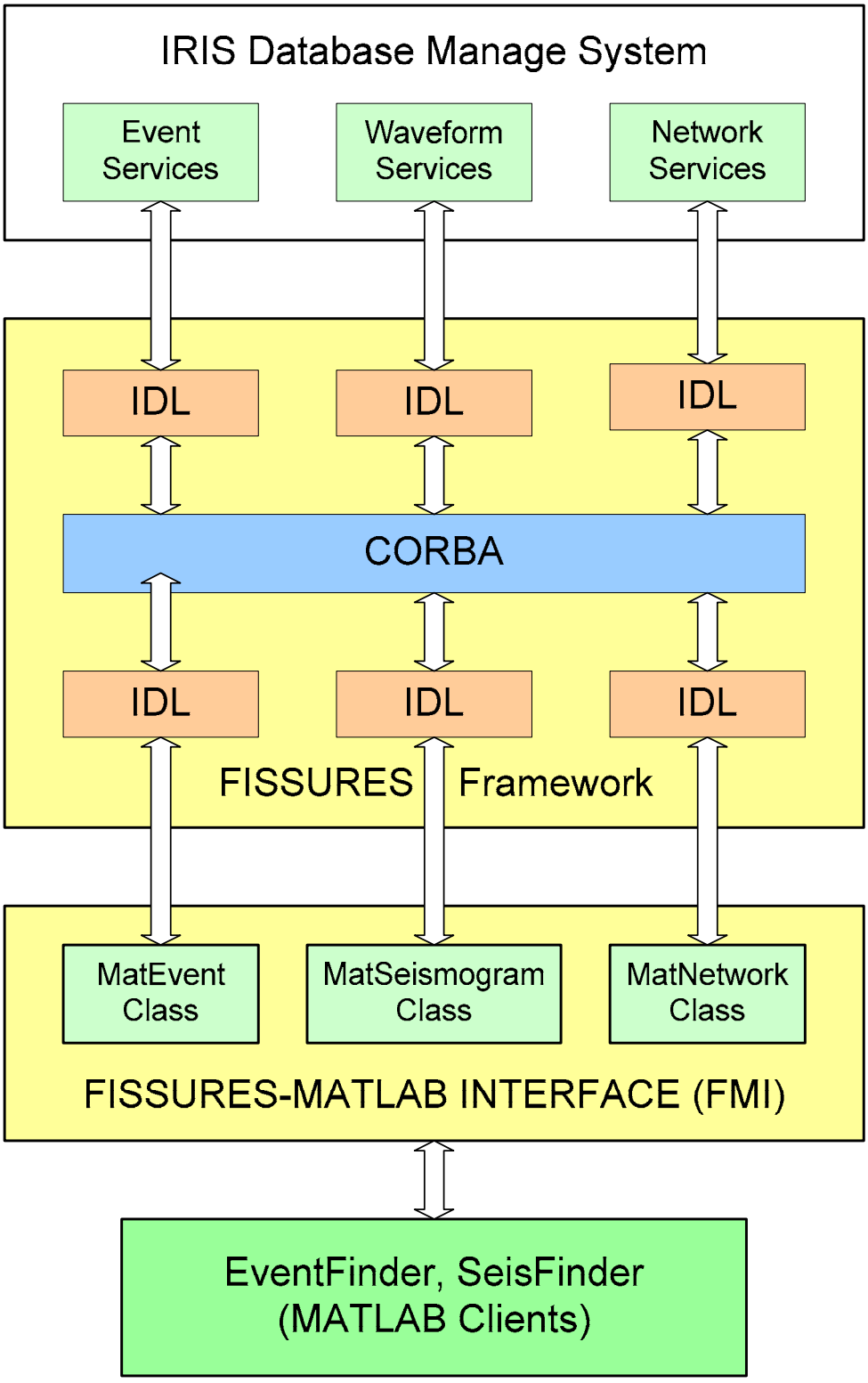

Figure 1. Overview of FISSURES and FMI

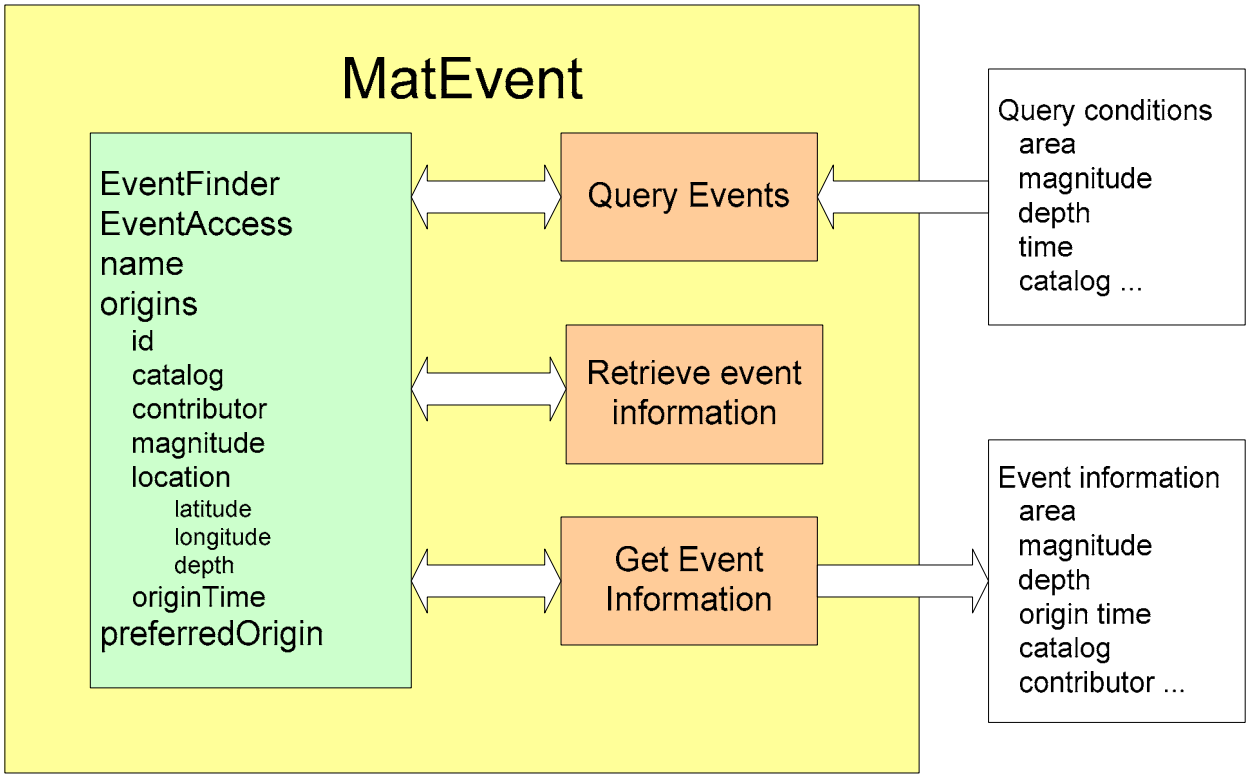

Figure 2. Structure of MatEvent class

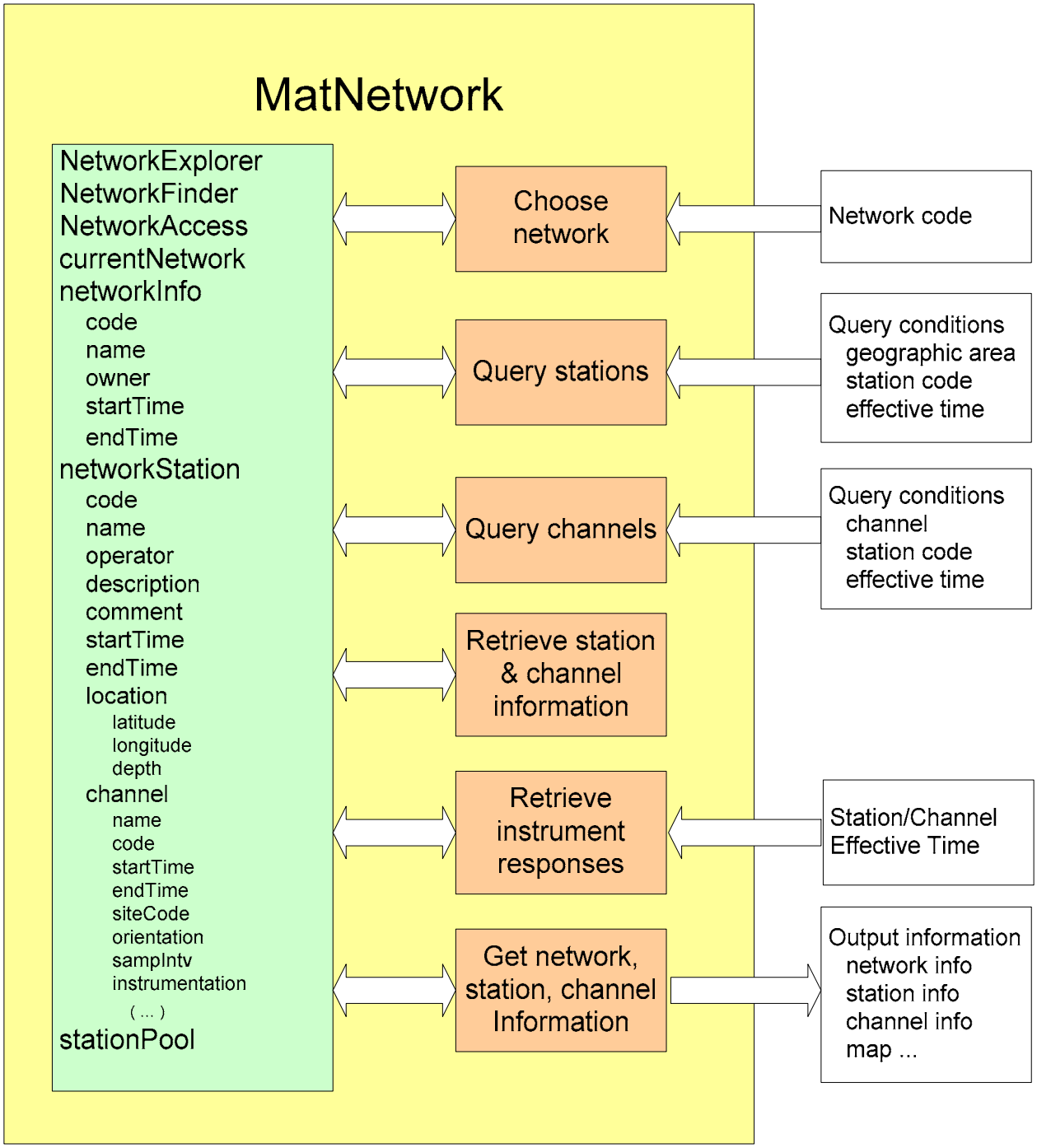

Figure 3. Structure of MatNetwork class

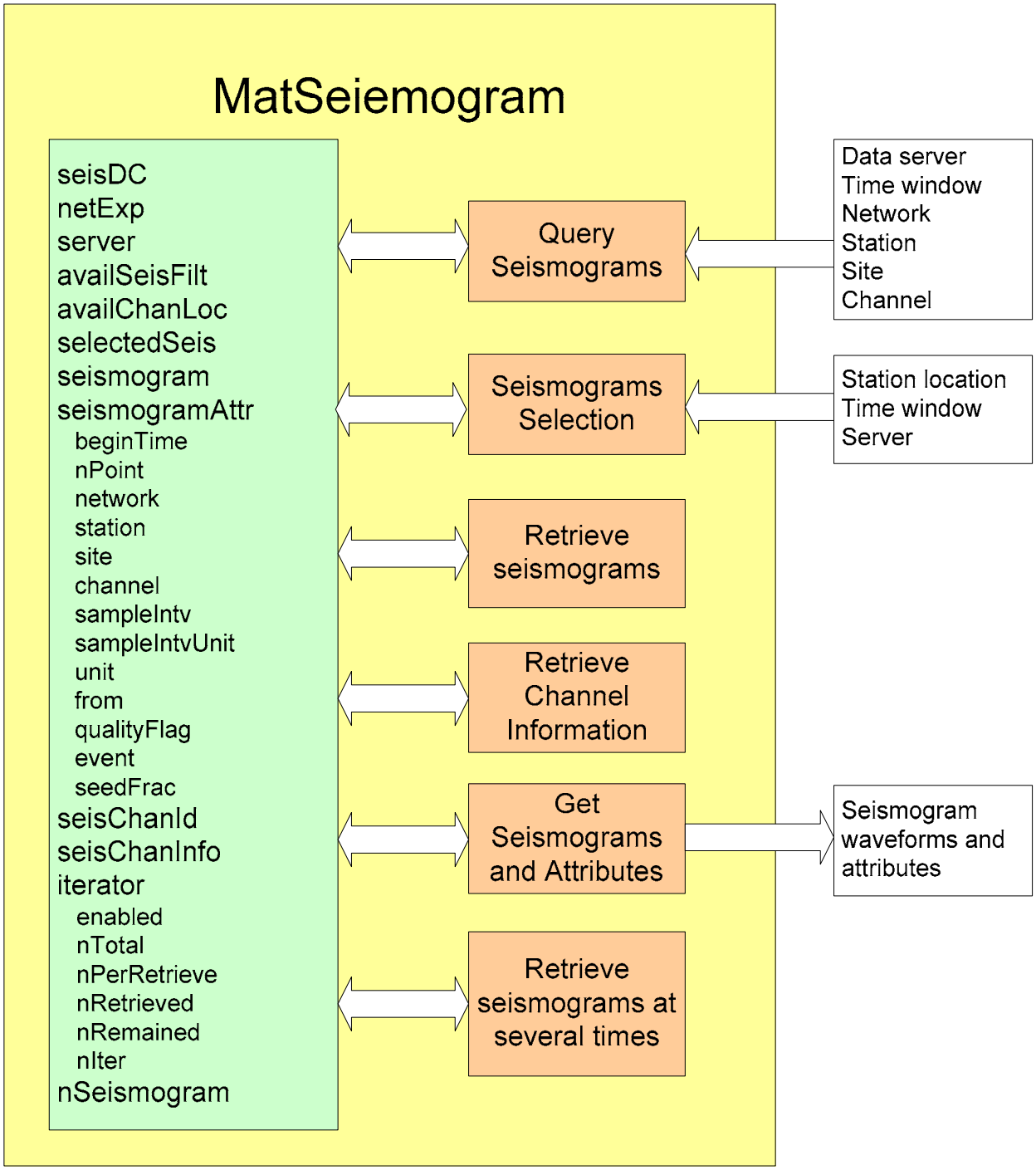

Figure 4. Structure of MatSeismogram class

FMI REFERENCES

1. Class MatEvent

Data members:

Name	Type	Description
`eventFnd`	Java object	Java object of class EventFinder
`eventAccess`	Java object	Java object of class EventAccess
`nEvent`	Double	Number of events
`name`	Cell array	Name of events
`origin`	Cell array	Origin
`preferredOrigin`	Structure array	Preferred Origins

Structure fields:

```
preferredOrigin(i).id                    (the i^th event)
                  .catalog
                  .contributor
                  .magnitude
                  .location
                  .originTime
```

If let `event_origin=origin{i}`, then `event_origin` is a structure array:

```
  event_origin(j).catalog            (the j^th origin for i^th the event)
                 .contributor
                 .magnitude
                 .location
                 .originTime

magnitude.value(k)
         .type{k}

location.latitude
        .longitude
        .depth
        .depthUnit
```

Methods:

Name	Description
`MatEvent`	Constructor of MatEvent
`find_event`	Query events based on given conditions
`retrieve_event_info`	Retrieve event information for found events
`list_event`	List event information
`map_event`	Map events

`known_catalog`	Get names of known catalogs
`known_contributor`	Get names of known contributors

Basic MATLAB methods:

Name	Description
`display`	Display MatEvent information
`get`	Get data from the object
`subsref`	Allow indexed and subscripted reference (see below)

Subscripted reference: (The index is optional)

```
myEvent.nEvent
        .name{i}
        .origin{i}
        .preferredOrigin(i)
        .preferredOrigin(i).magnitude
                           .location
                           .originTime
                           .catalog
                           .contributor
```

How to use MatEvent:

1. Open CORBA connection to IRIS DMC
   ```
   open_orb
   ```

2. Create and initialize an object of `MatNetwork` and initialize it
   ```
   myEvent=MatEvent(orb);
   ```

3. Query events for given conditions. The conditions could be one or combination of area, depth range, time range, magnitude range, search type, catalog, or contributor. The query does not return detail information of events, only event accesses.
   ```
   myEvent=find_event(myEvent, 'condition1', value1,
   'condition2', value2, … );
   ```

4. Retrieve event information from DMC
   ```
   myEvent=retrieve_event_info(myEvent);
   ```

5. Show event information. You can list event information on screen or plot event locations on map.
   ```
   list_event(myEvent);
   map_event(myEvent, MapTitle);
   ```

2. Class MatNetwork

Data members:

Name	Type	Description
networkFnd	Java object	Java object of NetworkFinder
networkExp	Java object	Java object of NetworkExplorer
networkInfo	Structure array	Network information
networkAccess	Java object	Java object of NetworkAccess
networkNum	Double	Number of networks
chosenNetwork	Double array	Chosen networks
currentNetwork	Double	Index of current chosen network
networkStation	Structure array	Stations for current chosen network
networkStationNum	Double	Number of stations in current network
stationPool	Structure array	Stations in station pool
stationPoolNum	Double	Number of stations in station pool
findStationDone	Boolean	Done finding stations
stationAddedToPool	Boolean	Done adding stations to station pool

Structure fields:

```
networkInfo.code
            .name
            .owner
            .description
            .startTime
            .endTime

networkStation.code      (same for structure stationPool)
              .name
              .networkCode
              .operator
              .description
              .comment
              .startTime
              .endTime
              .location.latitude
                       .longitude
                       .elevation
              .networkIndex
              .channel.name
                      .code
                      .startTime
                      .endTime
                      .siteCode
                      .orientation.azimuth
                                  .dip
                      .sampIntv
```

```
                    .instrumentation  (see detail below)


instrumentation.effectiveTime.startTime
                              .endTime
               .clockType
               .sensor.lowFreq
               .highFreq
               .sensitivity.frequency
               .factor
               .stageNum
               .stages.type
                      .inputUnit
                      .outputUnit
                      .gainFreq
                      .gainFactor
                      .normalizationFreq
                      .normalizationFactor
                      .decimation.factor
                                 .offset
                                 .inputRate
                      .filters.type
     (following 4 fields are for COEFFICIENT filters)
                               .coeff_B
                               .coeffError_B
                               .coeff_A
                               .coeffError_B
     (following 4 fields are for POLEZERO filters)
                               .poles
                               .poleErrors
                               .zeros
                               .zeroErrors
     (following 6 fields are for LIST filters)
                               .amplitude
                               .amplitudeError
                               .frequency
                               .frequencyUnit
                               .phase
                               .phaseError
                               .phaseUnit


(Stage type could be ANALOG, DIGITAL, COMPOSITE, or LAPLACE).
(Filter type could be COEFFICIENT, POLEZERO, or LIST)
```

Methods:

Name	Description
`MatNetwork`	Constructor of MatNetwork
`retrieve_network`	Retrieve network accesses and information from DMC
`choose_network`	Choose network(s)
`list_network`	List network information
`set_current_network`	Set current network
`find_station`	Query stations for current network
`find_station_channel`	Query channels for network stations
`find_channel`	Query channels (not implemented yet)
`find_channel_by_code`	Query channels by station and channel code
`station_pool_add`	Add stations to station pool
`station_pool_clear`	Clear station pool
`list_station`	List station information
`list_channel`	List channel information for specified station
`map_station`	Map stations
`get_response`	Retrieve instrumentations for specified channels

MATLAB basic methods:

Name	Description
`display`	Class MatNetwork display method
`subsref`	Allow indexed and subscripted reference (see below)

Indexed and subscripted reference: (The index is optional)

```
myNetworkDC.networkNum
           .networkInfo(i)
           .networkStationNum
           .networkStation(i)
           .stationPoolNum
           .stationPool(i)
           .currentNetwork
           .chosenNetwork
```

How to use MatNetwork:

1. Open CORBA connection to DMC
   ```
   open_orb
   ```

2. Create an object of `MatNetwork` and initialize it
   ```
   myNet=MatNetwork(orb);
   myNet=retrieve_network(myNet);
   ```

3. Choose networks you are interested in. You can choose multiple networks at one time by name, code, start time, and/or available time.

```
myNet=choose_network(myNet, 'condition1', value1,
'condition2', value2, …)
```

4. Set current network. You can only query stations and channels for one network at a time. The current network is determined by index, which is saved in data member `currentNetwork`.
```
chosenNet=myNet.choseNetwork;
myNet=set_current_network(myNet,chosenNet(1));
```

5. Query stations and channels. There are two ways. One way is to query stations first, limiting search by list of stations using wildcards, and by geographic area. Then search theses stations for channel codes limited by wildcards and by availability dates. The other way is to query channels given station code and channel code, and channels that satisfy conditions are grouped and ordered by station. In either case, you can use available time condition on queries to ensure the stations and channels found are effective at the specified time.

   a. Query stations first, then channels:
```
myNet=find_station(myNet, 'condition1', value1,
'condition2', value2, …)
myNet=find_station_channel(myNet, 'channel code',
'available time');
```

   b. Query channels by code and grouped by stations:
```
myNet=find_channel_by_code(myNet, 'station code', 'channel
code', 'available time')
```

6. Save query results. The results of query are stored in data member `networkStation`. If you set another network, or do another query, the contents of `networkStation` will be cleared. You can use `station_pool_add()` to save current query results into data member `stationPool` for further use. If you want, you can use `station_pool_clear()` to clear station pool.
```
myNet=station_pool_add(myNet);
myNet=station_pool_clear(myNet);
```

7. Show station and channel information. You can list station information on screen or plot station locations on a map. You also can list channel information for station specified by code or index.
```
list_station(myNet,option);
map_station(myNet,option); or
map_station(myNet,option,'label');

list_channel('station_code',option); or
list_channel(station_index,option);
```

`option` could be `'NetworkStation'` or `'StationPool'`, which determines which station information to be shown

8. Retrieve instrument responses for specified channels at a given time. You can specify stations and channels by their indices or names. However, only stations and channels already saved in `networkStation` will be searched. The results will be saved into `networkStation`. Only the indices of stations and channels whose instrument responses are successfully retrieved will be returned.

```
[myNet, staChanIndex] = get_response(myNet,'BHW','all','2000-01-
   01T00:00:00.0Z');
[myNet, staChanIndex] = get_response(myNet,[1:3],0);
sta = myNet.networkStation(staChanIndex(1).staIndex);
instru = sta.channel(staChanIndex(1).chanIndex(1)).instrumentation;
```

3. Class MatSeismogram

Data members:

Name	Type	Description
seisDc	Java object	Java object of class DataCenter
netExp	Java object	Java object of class NetworkExplorer
server	String	Name of the seismogram server
availSeisFilt	Java object array	Java object array of class RequestFilter
availChanLoc	Structure array	Channel locations for available seismograms
selectedSeis	Integer array	Index of selected seismograms
nSeismogram	Integer	Retrieved number of seismograms
seismogram	Cell array	Retrieved seismograms (data vector)
seismogramAttr	Structure array	Retrieved seismogram attributes
seisChanId	Java object array	Java object array of class ChannelId
seisChanInfo	Structure array	Channel information for retrieved seismograms
iterator	Structure	An iterator to control multiple retrieving

Structure fields:

```
availChanLoc(i).name
               .longitude
               .latitude
               .elevation
               .elevationUnit

seismogramAttr(i).beginTime
                 .nPoint
                 .network
                 .station
                 .site
                 .channel
                 .sampleIntv
                 .sampleIntvUnit
                 .unit
                 .from
                 .qualityFlag
                 .event
                 .seedFrac

seisChanInfo(i).name
               .networkCode
               .stationCode
               .citeCode
               .channelCode
               .samplingIntv
               .samplingIntvUnit
```

```
                .location.logitude
                         .latitude
                         .elevation
                         .elevationUnit
                .orientation.azimuth
                            .dip
                .startTime
                .endTime

iterator.enabled
        .nTotal
        .nPerRetrieve
        .nRetrieved
        .nRemained
        .nIter
```

Methods:

Name	Description
`MatSeismogram`	Constructor of `MatSeismogram`
`seis_query`	Search for available seismograms
`channel_location`	Retrieve channel locations of available seismograms
`seis_select`	Select a subset of available seismograms
`seis_retrieve`	Retrieve seismograms and attributes
`channel_info`	Retrieve channel information for retrieved seismograms
`get`	Get data from MatSeismogram object
`enable_iterator`	Enable the iterator for multiple retrieving
`disable-iterator`	Disable the iterator for multiple retrieving
`init_iterator`	Initialize the iterator for multiple retrieving

MATLAB basic methods:

Name	Description
`display`	Class MatNetwork display method
`get`	Get data from the object
`subsref`	Allow indexed and subscripted reference (see below)

Indexed and subscripted reference: (The index is optional)

```
    mySeis.nAvailSeis
          .availChanLoc(i)
          .availFrom(i)
          .nSelect
          .selectedSeis
          .nSeismogram
          .seismogram(i)
          .seismogramAttr(i)
          .seisChanInfo(i)
```

How to use MatSeismogram:

1 Open CORBA connection to DMC
```
open_orb
```

2 Create an object of `MatSeismogram` and specify the server name. Currently the available servers are Pond, Archive, and Bud. Please not if you choose Archive, the processing speed may be very slow.
```
mySeis = MatSeismogram(orb, 'Pond');
```

3 Search seismograms for given stations/channels and time window
```
mySeis = seis_query(mySeis,timeWindow,'I*',{'A*','RWW'},'','B**');
mySeis = seis_query(mySeis,timeWindow,'*','*','','*');
```

4 Retrieve channel locations of available seismograms
```
mySeis = channel_location(mySeis);
```

5 Select seismograms
```
[mySeis nSelect] = seis_select(mySeis, origin, chanArea, timeWindow);
```

6 Retrieve seismogram attributes
```
mySeis = seis_retrieve(mySeis,'selected');
```

7 Retrieve detailed channel information of retrieved seismograms
```
mySeis = channel_info(mySeis);
```

8 Get seismogram waveforms, attributes and channel information
```
seismogram = get(mySeis,'seismogram');
seisAttr = get(mySeis,'seismogramAttr');
seisChanInfo = get(mySeis,'seisChanInfo');
```

Multiple retrieving:
Sometimes, depending on the size of your computer's memory, the "out of memory" error may occur for retrieving a large number of seismograms. If this happens, you can break your retrieving job into small parts by enabling the built-in iterator.

1. Enable the iterator
```
mySeis = enable_iterator(mySeis);
```

2. Initialize the iterator to retrieve 20 seismograms at once
```
mySeis = init_iterator(mySeis,20);
```

3. Retrieve until all seismograms have been retrieved
```
nIter = 0;
nSeis = 0;
while(1)
    nIter = nIter + 1;

    % retrieve seismograms from the DMC
    [mySeis, nRetrieve] = seis_retrieve(mySeis,'selected');
    mySeis  = channel_info(mySeis);
    nSeis = nSeis + nRetrieve;
```

```
        % save seismograms into files
        ot = str2time(eventOrigin.originTime);
        ot(6) = round(ot(6));
        tempFileName = sprintf('%04d-%02d-%02dT%02d-%02d-%02d_%03d.mat', ot, nIter);
        seismogram = get(mySeis,'seismogram');
        seisAttr = get(mySeis,'seismogramAttr');
        seisChanInfo = get(mySeis,'seisChanInfo');
        eval(['save ' tempFileName ' seismogram seisAttr seisChanInfo']);

        iterator = get(mySeis,'iterator');
        if iterator.nRemained == 0
            break;
        end;
    end;
```

Memory Problems:

It is very difficult to give you a suggestion of the number of seismograms to retrieve at once, even if I know the physical memory size on your computer. Since MATLAB does not support reference or pointer like Java and C++ (to make MATLAB extremely easy to learn and use), the memory management of MATLAB is very inefficient. Every parameter in and out functions needs a fresh copy. There may be several copies of seismogram data in the memory when the problem is running. Therefore it is very difficulty to calculate how much memory left in runtime. On the other hand, the memory allocation depends on memory fragments. Sometimes the "out of memory" error does not occur until you use MATLAB for a while, and it may happen in either MATLAB code or Java code. MATLAB provides a memory optimization command called "pack". Unfortunately, the pack command does not work, if there exist Java objects in the workspace.

So here what I can suggest is, if the "out of memory" happens, to quite and restart MATLAB to see if the problem can be solved. If not, you can enable the iterator to retrieve seismograms by several times. You can try a reasonable number, say 100, to start. From my experience, it should be ok in most situations for systems with 512MB physical memory. If the problem still exists, then reduce the number by half and try again until the problem is solved.

4. FMI Tools

We provide several generic functions for convenience, which do not belong to any class.

1. `open_orb`: open CORBA connection to DMC. This is required for initialization of all classes.
2. `close_orb`: close CORBA connection.
3. `FMITime`: Description of time formats accepted in FMI
4. `time2str & str2time`: conversion between FISSURES time format (string) to MATLAB vector format. Both string and vector formats are accepted in all FMI methods.
5. `timedif`: calculate the time difference between two times
6. `timeshift`: calculate time with a shift
7. `timecmp`: compare two time instants
8. `local2UT`: convert local time to UTC
9. `phaseName`: Description of seismic phase naming rules
10. `coortr`: geocentric/geographic coordinate transformation
11. `delaz`: compute earthquake/station distance and azimuth
12. `FMI_ver`: return the FMI version number
13. `FMI_systemcheck`: check system environemts

Use "`help function_name`" for more detailed information on the generic functions and the methods for each class.