BioMOBY Exceptions Handling

Provider side prototype

Request for Comments

(Not official v2.0.3 December 2005)

Contributions:

Sergio Ramírez
Enrique de Andrés Saiz
Johan Karlsson
José-María Fernández
Antonio J. Pérez
Martin Senger
José Manuel Rodríguez
Coordinador: Oswaldo Trelles, David González-Pisano

National Bioinformatics Institute (INB)
Spain

Madrid, December 2005

1. - Preliminaries

This document contains exception reporting library which has been proposed by INB[1] to bioMoby (see 0501INB-ExceptionReporting-v2.0 documentation). The proposal was discussed at the INB Meeting in Málaga (July, 2005) with the participation of Martin Senger and Edward Kawas.

Perl's Built-In Exception Handling Mechanism

Perl has a built-in exception handling mechanism, the *eval {}* block. It is implemented by wrapping the code that needs to be executed around an eval block and the $@ variable is checked to see if an exception occurred. The typical syntax is:

```
eval {
  ...
};
if ($@) {
  errorHandler($@);
}
```

Within the eval block, if there is a syntax error or runtime error, or a die statement is executed, then an undefined value is returned by eval, and $@ is set to the error message. If there was no error, then $@ is guaranteed to be a null string.

What's wrong with this? Since the error message store in $@ is a simple scalar, checking the type of error that has occurred is error prone. Also, $@ doesn't tell us where the exception occurred. To overcome these issues, exception objects were incorporated in Perl 5.005.

```
eval {
    open(FILE, $file) ||
      die MyFileException->new("Unable to open file - $file");
  };

if ($@) {
    # now $@ contains the exception object of type MyFileException
    print $@->getErrorMessage();
        # where getErrorMessage() is a method in MyFileException class
  }
```

The exception class (MyFileException) can be built with as much functionality as desired. Therefore, it will be good way built apropiate Exception class to BioMOBY.

Also, it is important to say that is not necessary to use the $@ how exception “object”, we will be able to create new variable which will be built with Moby functionality. For example:

```
my ($exceptionObject);
eval {
    open(FILE, $file) ||
      die ($exceptionObject = MobyException->new();
  };
```

[1] Instituto Nacional de Bioinformática (INB), Spain

```
if ($@) {
    if ($exceptionObject->isa(‘MobyException’)) {
            print $exceptionObject->getErrorMessage();
  }
```

2. – Description of Exception class

The Exception class contains four attributes:

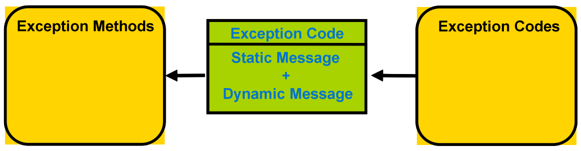
- **queryID** – mobyData identifier where is located the exception -.
- **refElement** – article Name of Moby input -.
- **code** – numeric identifier of exception -.
- **message** – description of exception -.
- **type** – type of exception (error, warning, or information) -.

Then, from attributes above we obtain the next methods:

- **new**
- **getExceptionCode**
- **getExceptionMessage**
- **getExceptionType**
- **setExceptionCode**
- **setExceptionMessage**
- **setExceptionType**
- **retrieveExceptionResponse**
- **retrieveEmptyMobyData**
- **retrieveEmptyMobySimple**
- **retrieveEmptyMobyCollection**
- **embedMOBYArticlesIntoMOBYData**

NOTE: I guess we could add new methods: new method that embeds the exception reporting into serviceNotes; or other one, which embeds MobyArticles (usually empty) into mobyData.

The Exception library is consisted of two files: **MobyException** and **MobyExceptionCodes**. The above methods are stored into MobyException file. The exception codes and descriptions are stored into ExceptionCodes file. That division help us to separate the exception data from the exception methods and then it will be easier the new insertion of exception codes and exception descriptions.



See than there are new concepts: ***static message*** and ***dynamic message***. The first one is standard exception description returned by exception library (see 3 section, Exception Codes) and dynamic message is description giving by service and then it grants more specification to exception message.

BioMOBY Perl libraries extension

The new MobyException library is composed by two files: MobyException and MobyExceptionCodes. The last one, it contains the exception "table" of codes and message. The first methos, contains the methods of MobyException "object". Here we show that methods:

New	
Name	New
Function	Instance a new *MobyException* object. When you "bless" MobyException class you have to be careful: the exception code has to belong to Exception table and type of exception has to be within range of values (error, warning, or info).
Usage	My $exceptionObject = MobyException->new(%args)

Args	%args = (
	queryID => "identifier where is located the exception", # '' by default
	refElement => "article Name of Moby input", # '' by default
	code => "numeric identifier of exception", # 0 by default
	message => "description of exception" # undef by default
	type => "type of exception" # undef by default
	)

Returns	MobyException "object"

getExceptionCode	
Name	getExceptionCode
Function	Returns code of MobyException "object". Default value of exception code is cero,that means, it has to set new code value.
Usage	My $excepCode = $exceptionObject->getExceptionCode();
Args	
Returns	0 => error or Integer (exceptionCode) => OK

getExceptionMessage	
Name	getExceptionMessage
Function	Returns description of Exception. The description is composed by two messages: standard and dynamic messages. The first one is given by MobyException library, the next one is given by user dynamically. The standard message depends on exception code that is stored in attribute of MobyException object. If the exception code does not exist, then this method returns undef.
Usage	My $excepMessage = $exceptionObject->getExceptionMessage();
Args	
Returns	Undef => error or String (Exception message) => OK

getExceptionType	
Name	getExceptionType
Function	Returns type of Exception. The types of exceptions are error, warning, or information. If the exception type does not exist, then this method returns undef.
Usage	My $excepType = $exceptionObject->getExceptionType();
Args	
Returns	Undef => error or String (Exception type: error, warning, or info) => OK

setExceptionCode	
Name	setExceptionCode
Function	Assign code to given MobyException “object”. The input code has to belong to table exception codes given by MOBY
Usage	$exceptionObject->setExceptionCode($excepCode);
Args	Integer (Exception Code) Actually the method does not check if input code is correct…So, be careful!!!!
Returns	

setExceptionMessage	
Name	setExceptionMessage
Function	Assign description of Exception to MobyException “object”. The message has to correspond to “dynamic message” giving by developer.
Usage	$exceptionObject->setExceptionMessage();
Args	String (dynamic exception message)
Returns	

setExceptionType	
Name	setExceptionType
Function	Assign type of Exception to MobyException “object”. The input type has to be within range of values: error, warning, or info.
Usage	$exceptionObject->setExceptionType();
Args	String (type of exception: error, warning or info)
Returns	

retrieveExceptionResponse	
Name	retrieveExceptionResponse
Function	Returns xml block of MobyException response. The kind of exception response which will be returned depending on MobyException object itself (error, warning or info). You have to be careful whether MobyException object has bad declared some attributes (code and type), then this method does not return good response.
Usage	My $errorResponse = $exceptionObject->retrieveExceptionResponse();
Args	
Returns	String (xml block of exception) => OK || error message => Error

retrieveEmptyMobyData	
Name	retrieveEmptyMobyData
Function	Returns xml block of empty MobyData response. For that, it will use the values of attributes of MobyException “object”
Usage	My $mobyResponse = $exceptionObject->retrieveEmptyMobyData();
Args	
Returns	String (xml block of empty MobyData response)

retrieveEmptyMobySimple	
Name	retrieveEmptyMobySimple
Function	Returns xml block of empty SIMPLE MobyArticle. For that, it will use given articleName input.
Usage	My $mobyResponse = $exceptionObject->retrieveEmptyMobySimple($artNameOutput);
Args	Article name of SIMPLE MOBYArticle output
Returns	String (xml block of empty SIMPLE MobyArticle)

retrieveEmptyMobyCollection

Name	retrieveEmptyMobyCollection
Function	Returns xml block of empty COLLECTION MobyArticle. For that, it will use given articleName input.
Usage	My $mobyResponse = $exceptionObject->retrieveEmptyMobyCollection($artNameOut);
Args	Article name of COLLECTION MOBYArticle output
Returns	String (xml block of empty COLLECTION MobyArticle)

embedMOBYArticlesIntoMOBYData

Name	embedMOBYArticlesIntoMOBYData
Function	Returns xml block of MOBYData. MOBYArticles given by input are add into output MOBYData.
Usage	My $mobyResponse = $exceptionObject-> embedMOBYArticlesIntoMOBYData ($MOBYArticles);
Args	String List of MOBYArticles
Returns	String (xml block of error MOBYData Response)

3. – Description of Exception codes

The following is a list describing the exception conditions, such as overflows and errors resulting from incorrect or unmatched data, which are generated during program execution. The error codes are compatible with the LSAE specification.

(a) New (BioMOBY specific) error types not included in LSAE specification

(b) New error types

(c) Error types on MOBYContent and MOBYData level

(d) There are specified exception message to secondary articles because it is necessary to identify the MOBYData article. If a secondary article produces an exception, there are not ways to know which articles do it.

(e) Error offering by developer: Paul Gordon

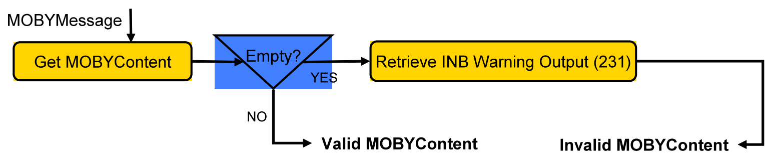
3.1 – Exception codes dealing with analysis data:

Code	Name	Description
200	UNKNOWN_NAME	Setting input data under a non-existing name, or asking for a result using an unknown name
201	INPUTS_INVALID	Input data are invalid; they do not match with their definitions, or with their dependency conditions[2]
202	INPUT_NOT_ACCEPTED	Used when a client tries to send input data to a job created in a previous call but the server does not any more accept input data
221[(a)]	INPUT_REQUIRED_PARAMETER	Service require parameter X
222[(a)]	INPUT_INCORRECT_PARAMETER	Incorrect parameter X
223[(a)]	INPUT_INCORRECT_SIMPLE	Incorrect input in simple article
224[(a)]	INPUT_INCORRECT_SIMPLENB	Service requires two or more simple articles
225[(a)]	INPUT_INCORRECT_COLLECTION	Incorrect input in collection article
226[(a)]	INPUT_EMPTY_OBJECT	Empty input object
231[(c)]	INPUT_EMPTY_MOBYCONTENT	Empty MOBYContent.
232[(c)]	INPUT_INCORRECT_MOBYDATA	Malformed MOBYData. For example, there is not queryID.
233[(c)]	INPUT_EMPTY_MOBYDATA	Empty MOBYData. There is not article List.

[2] Taken from LSAE, in BioMOBY this means a generic invalid input error. Other specific invalid input errors listed below

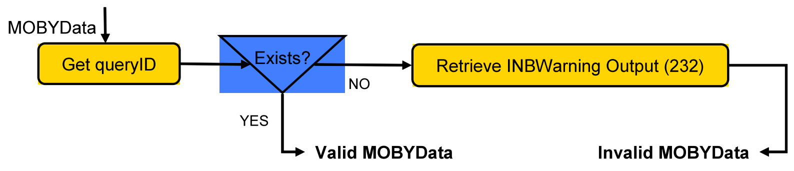
- MOBYContent level's checking:

One of the first validations that MOBYServer has to check is whether MOBYContent, which is sent to MOBYServer, is empty or not. In that case, MOBYServer has to retrieve **warning** exception (and nothing more).
For this type of checking it is recommended to use **retrieveWarningException** method.



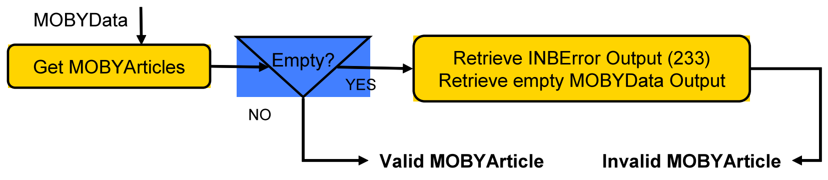
- MOBYData level's checking:

For each MOBYData, which is inside of MOBYContent, has to be checked. If its queryID attribute does not exist; then service will retrieve **warning** exception and it will continue the normal process knowing that MOBYData output will not be able to have queryID.
For this type of checking it is recommended to use **retrieveWarningException** method.



- MOBYArticle level's checking:

Also, it is necessary to check if the list of MOBYArticles, which is within of MOBYData, exists. It means, MOBYData is empty or not. In that case, service will return **error** exception message and empty MOBYData output giving queryID.
It is recommended to use **retrieveErrorException** method.



3.2 – Exception codes dealing with analysis execution:

Code	Name	Description
300	NOT_RUNNABLE	The same job has already been executed, or the data that had been set previously do not exist or are not accessible anymore. Life Sciences Analysis Engine Adopted Specification
301	NOT_RUNNING	A job has not yet been started. Note that this exception is not raised when the job has been already finished.
302	NOT_TERMINATED	A job is not interruptible for some reason.

3.3 – Error codes dealing with analysis metadata:

Code	Name	Description
400	NO_METADATA_AVAILABLE	There are no metadata available

3.4 – Error codes dealing with notification:

Code	Name	Description
500	PROTOCOLS_UNACCEPTED	Used when a server does not agree on using any of the proposed notification protocols

3.5 – General error codes:

Code	Name	Description
600	INTERNAL_PROCESSING_ERROR	A generic catch-all for errors not specifically mentioned elsewhere in this list
601	COMMUNICATION_FAILURE	A generic network failure
602	UNKNOWN_STATE	Used when a network call expects to find an existing state but failed. An example is an unknown handler representing a Job (unknown Job_ID, typical for WebServices platform)
603	NOT_IMPLEMENTED	A requested method is not implemented. Note that the method in question must exist (otherwise it may be caught already by the underlying protocol and reported differently) - but it has no implementation
621[b]	SERIVCE_NOT_AVAILABLE	Service not available

3.6 – Service intrinsic errors:

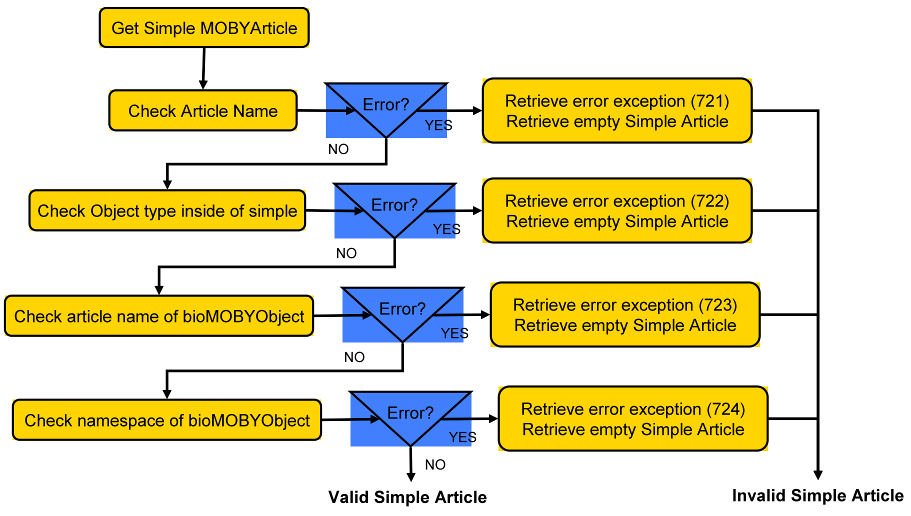
Code	Name	Description
700[e]	OK	Everything was ok
701[a]	SERVICE_INTERNAL_ERROR	Specific errors from the BioMOBY service[3]
702[a]	OBJECT_NOT_FOUND	Object not found with the given input[4]
703[e]	DATA_NOT_LONGER_VALID	A sequence indentifier that has been retracted
704[e]	INPUT_BIOLOGICALLY_INVALID	The input does not make sense biologically
705[e]	DATA_TRANSFORMED	E.g. non-DNA chars ignored in DNA search
721[b]	INCORRECT_ARTICLE_NAME	The specified name of MOBYData article (simple or collection) is wrong or does not exist.
722[b]	INCORRECT_OBJECT_TYPE	Incorrect object data type from specified MOBYData article (simple or collection)
723[b]	INCORRECT_ARTICLENAME_OBJECT	The specified article name of BioMOBY Object is wrong.
724[b]	INCORRECT_NAMESPACE_OBJECT	The namespace of specified BioMOBY Object is invalid
731[d]	INCORRECT_ARTICLENAME_SECONDARY	The specified name of secondary is wrong or does not exist.
732[d]	INCORRECT_DATA_TYPE_SECONDARY	Incorrect data type from specified secondary article
733[d]	INCORRECT_VALUE_SECONDARY	The value of secondary article is invalid. It is not inside of correct range.
734[d]	INCORRECT_VALUE_AND_DEFAULTVALUE_SECONDARY	There is not SECONDARY value and registered SECONDARY article has not default value

- SIMPLE MOBYArticle checking:

From one simple MOBYArticle we check several values that are embed within it (see flowchart below). For this kind of checking it is recommended to use **retrieveErrorExceptionUsingSimple** method.

```
<Simple articleName="articleName"  (721) >
   <CommentedAASequence(722)   articleName="articleName"(723)   namespace="GenBank/GI"(724)  id="163483">
          <Integer namespace="primative" id="" articleName="Length">375</Integer>
          <String namespace="primative" id="" articleName="SequenceString">
          ATTGCGCATGCGAGCTAGTAGCATGCGATGAGGTCGATGCATCT
          </String>
          <String namespace="primative" id="" articleName="Description">B.taurus  prepreproelastase I mRNA</String>
    </ CommentedAASequence >
</Simple>
```

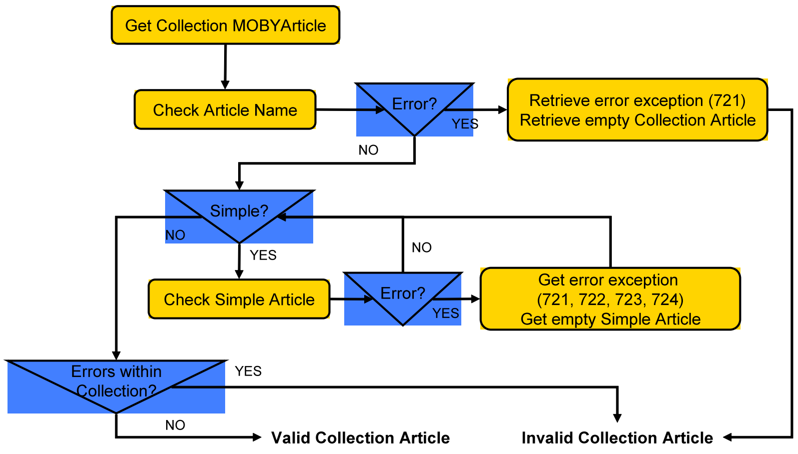
[3] I.e. from Blast 'No hits found' or 'Check the sequence format; it does not seem to be a nucleotide/Amino acid sequence'
[4] I.e. the specified namespace is wrong or does not exist (namespace supplied "SwissProt"; expected "Swiss-Prot")


Get Simple MOBYArticle
Check Article Name
Error?
YES
NO
Retrieve error exception (721)
Retrieve empty Simple Article
Check Object type inside of simple
Error?
YES
NO
Retrieve error exception (722)
Retrieve empty Simple Article
Check article name of bioMOBYObject
Error?
YES
NO
Retrieve error exception (723)
Retrieve empty Simple Article
Check namespace of bioMOBYObject
Error?
YES
NO
Retrieve error exception (724)
Retrieve empty Simple Article
Valid Simple Article
Invalid Simple Article

- COLLECTION MOBYArticle checking:

The next flow chart shows us how to check one collection article. Here you can see possible error codes.
retrieveErrorExceptionUsingCollection method is recommended to be used.

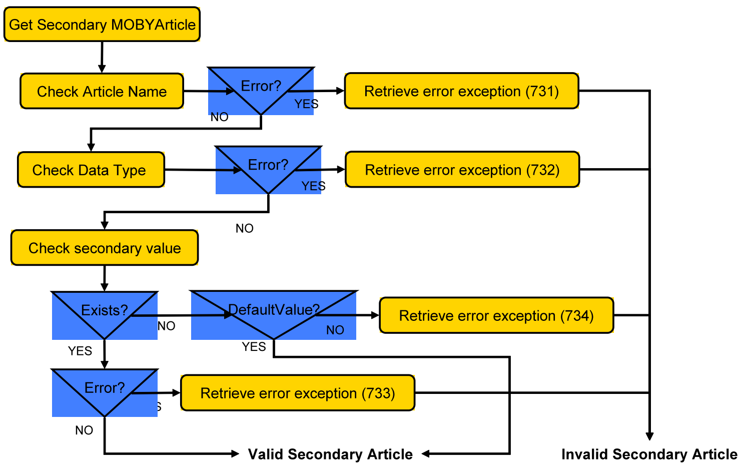
```
<Collection articleName="COLLECTIONarticleName"  (721) >
      <Simple articleName="SIMPLEarticleName1"  (721) >
            <CommentedAASequence(722)  articleName="articleName"(723)  namespace="GenBank/GI"(724)  id="163483">
                  <Integer namespace="primative" id="" articleName="Length">375</Integer>
                  <String namespace="primative" id="" articleName="SequenceString">
                        ATTGCGCATGCGAGCTAGTAGCATGCGATGAGGTCGATGCATCT</String>
                  <String namespace="primative" id="" articleName="Description">prepreproelastase I mRNA</String>
            </ CommentedAASequence >
      </Simple>
                                                …
      <Simple articleName="SIMPLEarticleNameN"  (721) > <!—bioMOBYobject -- > </Simple>
</Collection>
```



NOTE: See we use the same exception code when there is one error in article name as simple as collection. It is interesting to create new type for each exception but actually bioMOBY does not consider simple article name within collection.

- SECONDARY MOBYArticle checking:

Finally, we show how to check secondary article. You have to see that there is not empty MOBYData when secondary article has an error. Therefore, the client will not be able to know which secondary article is wrong. Then, it is important a good exception description; it is means, the exception message has to have the name of secondary article.
For this kind of checking it is recommended to use **retrieveErrorException** method.



Get Secondary MOBYArticle
Check Article Name
Error?
YES
NO
Retrieve error exception (731)
Check Data Type
Error?
YES
NO
Retrieve error exception (732)
Check secondary value
Exists?
NO
YES
DefaultValue?
NO
YES
Retrieve error exception (734)
Error?
NO
Retrieve error exception (733)
Valid Secondary Article
Invalid Secondary Article

3.7 – Client-side detected errors:

Client-side detected errors		
Code	**Name**	**Description**
800[(a)]	SERVICE_UNAVAILABE	Service Unreachable or not available
801[(a)]	QUOTA_EXCEEDED	Quota exceeded
802[(a)]	NO_ACCESS_ALLOWED	No access permissions
803[(a)]	SERVICE_TIMEOUT	Service timeout

4. – Scripts of MOBYException library:

The next we show the two scripts that compose the MOBYException library: MobyException.pm and MobyExceptionCodes.pm. Also, it is included test script which shows us how to use the MobyException class.

- **MobyException File:**

```
# Name of the package
package MobyException;

# Perl pragma to restrict unsafe constructs
use strict;

# Issue warnings about suspicious programming.
use warnings;

use Carp qw(croak);

use MobyExceptionCodes;

###############
# Constructor #
###############

sub new {
        # Get parameters
        my ($caller, %args) = @_;

        my ($caller_is_obj) = ref($caller);
        my ($class) = $caller_is_obj || $caller;
        my ($self) = {};

        # Add attributes to Exception "Object" if they are defined. Otherwise, default
values are added into.
        $self->{queryID} = (exists($args{queryID}) && defined($args{queryID})) ?
$args{queryID} : '';
        $self->{refElement} = (exists($args{refElement}) && defined($args{refElement})) ?
$args{refElement} : '';
        $self->{code} = (exists($args{code}) && defined($args{code})) ? $args{code} : 0;
        $self->{message} = (exists($args{message}) && defined($args{message})) ?
$args{message} : undef;

#       $self->{type} = (exists($args{type}) && defined($args{type})) ?  $args{type} :
undef; #  (undef | error | warning | info)
#       $self->{level} = (exists($args{level}) && defined($args{level})) ?  $args{level}
: undef; # (undef | mobySimple | mobyCollection | mobyData)

        # The magic object creation command
        bless ($self, $class);

        # Returns a new 'MobyException' object
        return $self;
        #die;
}

#################
# Method bodies #
#################

# Return exception code
sub getExceptionCode {
        # Get parameters
        my($self)=shift;

        croak("This is an instance method!")  unless(ref($self));

        return $self->{code};
}

# Return exception message
```

```
sub getExceptionMessage {
        # Get parameters
        my($self)=shift;
        my ($exceptionMessage) = undef;

        croak("This is an instance method!")  unless(ref($self));

        # Get standard exception message from given code
        my ($standardMessage) = MobyExceptionCodes::getExceptionCodeDescription($self-
>{code});

        # If standard message is not defined that means, the exception code is wrong =>
return undef
        return undef unless(defined($standardMessage));

        # User could add dynamic message into satandard exception message
        ($exceptionMessage) = (defined($self->{message})) ? $standardMessage.$self-
>{message} : $standardMessage;

        return $exceptionMessage;
}

# Assign exception code
sub setExceptionCode {
        # Get parameters
        my($self, $code) = @_;

        croak("This is an instance method!")  unless(ref($self));

        if (defined($code)) {
                $self->{code} = $code;
        } else {
                croak("input argument not defined");
        }
}

# Assign exception message
sub setExceptionMessage {
        # Get parameters
        my($self, $message) = @_;

        croak("This is an instance method!")  unless(ref($self));

        if (defined($message)) {
                $self->{message} = $message;
        } else {
                croak("input argument not defined");
        }
}

# Return xml block of ERROR exception response
# If the exception code is wrong, then the method returns undef
sub retrieveErrorExceptionResponse {
        # Get parameters
        my($self)=shift;
        my ($exceptionResponse) = undef;

        croak("This is an instance method!")  unless(ref($self));

        # Get standard exception message from given code
        my ($standardMessage) = MobyExceptionCodes::getExceptionCodeDescription($self-
>{code});

        # If standard message is not defined that means, the exception code is wrong =>
return undef
        return undef unless(defined($standardMessage));

        # User could add dynamic message into satandard exception message
        my ($exceptionMessage) = (defined($self->{message})) ? $standardMessage.$self-
>{message} : $standardMessage;

        return "<mobyException refQueryID='".$self->{queryID}."' refElement='".$self-
>{refElement}."' severity='error'>\n\t<exceptionCode>".$self-
>{code}."</exceptionCode>\n\t<exceptionMessage>".$exceptionMessage."</exceptionMessage>\
n</mobyException>";
}
```

```
# Return xml block of WARNING exception response
# If the exception code is wrong, then the method returns undef
sub retrieveWarningExceptionResponse {
	# Get parameters
	my($self)=shift;
	my ($exceptionResponse) = undef;

	croak("This is an instance method!")  unless(ref($self));

	# Get standard exception message from given code
	my ($standardMessage) = MobyExceptionCodes::getExceptionCodeDescription($self-
>{code});

	# If standard message is not defined that means, the exception code is wrong =>
return undef
	return undef unless(defined($standardMessage));

	# User could add dynamic message into satandard exception message
	my ($exceptionMessage) = (defined($self->{message})) ? $standardMessage.$self-
>{message} : $standardMessage;

	return "<mobyException refQueryID='".$self->{queryID}."' refElement='".$self-
>{refElement}."' severity='warning'>\n\t<exceptionCode>".$self-
>{code}."</exceptionCode>\n\t<exceptionMessage>".$exceptionMessage."</exceptionMessage>\
n</mobyException>";
}

# Return xml block of INFORMATION exception response
# If the exception code is wrong, then the method returns undef
sub retrieveInfoExceptionResponse {
	# Get parameters
	my($self)=shift;
	my ($exceptionResponse) = undef;

	croak("This is an instance method!")  unless(ref($self));

	# Get standard exception message from given code
	my ($standardMessage) = MobyExceptionCodes::getExceptionCodeDescription($self-
>{code});

	# If standard message is not defined that means, the exception code is wrong =>
return undef
	return undef unless(defined($standardMessage));

	# User could add dynamic message into satandard exception message
	my ($exceptionMessage) = (defined($self->{message})) ? $standardMessage.$self-
>{message} : $standardMessage;

	return "<mobyException refQueryID='".$self->{queryID}."' refElement='".$self-
>{refElement}."' severity='information'>\n\t<exceptionCode>".$self-
>{code}."</exceptionCode>\n\t<exceptionMessage>".$exceptionMessage."</exceptionMessage>\
n</mobyException>";
}

# Return xml block of one empty MobyData
sub retrieveEmptyMobyData {
	# Get parameters
	my($self)=shift;

	croak("This is an instance method!")  unless(ref($self));

	return "<moby:Data moby:queryID='".$self->{queryID}."' />";
}

# Return xml block of one empty simple MobyArticle
sub retrieveEmptyMobySimple {
	# Get parameters
	my($self)=shift;

	croak("This is an instance method!")  unless(ref($self));

	return "<moby:Simple moby:articleName='".$self->{refElement}."' />";
}

# Return xml block of one empty collection MobyArticle
sub retrieveEmptyMobyCollection {
	# Get parameters
```

```
        my($self)=shift;

        croak("This is an instance method!")  unless(ref($self));

        return "<moby:Collection moby:articleName='".$self->{refElement}."' />";
}

sub DESTROY {}

1;
```

- **MobyExceptionCodes File:**

```
#!/usr/local/bin/perl -w

package MobyExceptionCodes;

use strict;


##############################################################################
# PROTOTYPES
##############################################################################
sub getExceptionCodeDescription($);


#############################################################################
# NAME: getExceptionCodeDescription
#
# DESCRIPTION: An exception code is received by giving input and then,
#                       the   method   retrieves   itself   exception   code   and   exception
description.
#                       Also, if exception message input is defined, then it will be added
into
#                       exception description output.
#                       The exception error come from INB Exception Codes.
#                       See. 0501INB-ExceptionReporting-v1.7 documentation
#
# INPUTS:       - Exception Code
#                       - Dynamic Exception Description
#
# OUTPUTS:      - INB Exception hash: Code, Description
#
# MODIFIED DATE: 02-Sep-2005
#
# AUTHOR: Jose Manuel Rodriguez Carrasco -jmrc@cnb.uam.es- (INB-CNB)
##############################################################################
sub getExceptionCodeDescription($) {

                my ($exceptionCode) = @_;
                my ($INB_Exception) = undef;

switch: {

# ERROR CODES DEALING WITH ANALYSIS DATA
                if  ($exceptionCode == 200)  {  # UNKNOWN  NAME:  [200]  "Setting  input  data
under a non-existing name, or asking for a result using an unknown name"

                        ($INB_Exception) = {
                                'code' => $exceptionCode,
                                'message' => "Setting input data under a non-existing name,
or asking for a result using an unknown name.",
                        };

                } elsif ($exceptionCode == 201) { # INPUTS INVALID: [201] "Input data are
invalid; not match with its definitions, or with its dependency condition"

                        ($INB_Exception) = {
                                'code' => $exceptionCode,
                                'message'  =>  "Input  data  are  invalid;  not  match  with  its
definitions, or with its dependency condition.",
                        };

                } elsif ($exceptionCode == 202) { # INPUT NOT ACCEPTED: [202] "Input data
are accepted"

                        ($INB_Exception) = {
                                'code' => $exceptionCode,
                                'message' => "Input data are accepted.",
                        };

                }  elsif  ($exceptionCode  ==  221)  {  #  INPUT  REQUIRED  PARAMETER:  [221]
"Service require parameter X"

                        ($INB_Exception) = {
                                'code' => $exceptionCode,
                                'message' => "Service require parameter.",
                        };

                } elsif ($exceptionCode == 222) { # INPUT INCORRECT PARAMETER: [222]
"Incorrect parameter X"

                        ($INB_Exception) = {
                                'code' => $exceptionCode,
                                'message' => "Incorrect parameter.",
                        };

                } elsif ($exceptionCode == 223) { # INPUT INCORRECT SIMPLE: [223]
"Incorrect input in simple article"

                        ($INB_Exception) = {
                                'code' => $exceptionCode,
                                'message' => "Incorrect input in simple article.",
                        };

                } elsif ($exceptionCode == 224) { # INPUT INCORRECT SIMPLENB: [224]
"Service requires two or more simple articles"

                        ($INB_Exception) = {
                                'code' => $exceptionCode,
                                'message' => "Service requires two or more simple
articles.",
                        };

                } elsif ($exceptionCode == 225) { # INPUT INCORRECT COLLECTION: [225]
"Incorrect input in collection article"

                        ($INB_Exception) = {
                                'code' => $exceptionCode,
                                'message' => "Incorrect input in collection article.",
                        };

                } elsif ($exceptionCode == 226) { # INPUT EMPTY OBJECT: [226] "Empty
input object"

                        ($INB_Exception) = {
                                'code' => $exceptionCode,
                                'message' => "Empty input object.",
                        };

                } elsif ($exceptionCode == 231) { # INPUT EMPTY MOBYCONTENT: [231] "Empty
MOBYContent"

                        ($INB_Exception) = {
                                'code' => $exceptionCode,
                                'message' => "Empty MOBYContent.",
                        };

                } elsif ($exceptionCode == 232) { # INPUT EMPTY MOBYCONTENT: [232]
"QueryID does not exists"

                        ($INB_Exception) = {
                                'code' => $exceptionCode,
                                'message' => "QueryID does not exists.",
                        };

                } elsif ($exceptionCode == 233) { # INPUT EMPTY MOBYDATA: [233] "Empty
MOBYData"

                        ($INB_Exception) = {
                                'code' => $exceptionCode,
                                'message' => "Empty MOBYData.",
                        };

# EXCEPTION CODES DEALING WITH ANALYSIS EXECUTION
                } elsif ($exceptionCode == 300) { # NOT RUNNABLE: [300] "The same job has
already been executed, or the data that had been set previously do not exist or are not
accessible anymore"

                        ($INB_Exception) = {
                                'code' => $exceptionCode,
                                'message' => "The same job has already been executed, or
the data that had been set previously do not exist or are not accessible anymore.",
                        };
```

```
		} elsif ($exceptionCode == 301) { # NOT RUNNING: [301] "A job has not yet
been started"

			($INB_Exception) = {
				'code' => $exceptionCode,
				'message' => "The job has not yet been started.",
			};

		} elsif ($exceptionCode == 302) { # NOT TERMINATED: [302] "A job is not
interruptible for some reason"

			($INB_Exception) = {
				'code' => $exceptionCode,
				'message' => "The job is not interruptible for some
reason.",
			};

# EXCEPTION CODES DEALING WITH ANALYSIS EXECUTION
		} elsif ($exceptionCode == 400) { # NO METADATA AVAILABLE: [400] "There
are no metadata available"

			($INB_Exception) = {
				'code' => $exceptionCode,
				'message' => "There are no metadata available.",
			};

# EXCEPTION CODES DEALING WITH NOTIFICATION
		} elsif ($exceptionCode == 500) { # PROTOCOLS UNACCEPTED: [500] "Server
does not agree on using any of the proposed notification protocols"

			($INB_Exception) = {
				'code' => $exceptionCode,
				'message' => "Server does not agree on using any of the
proposed notification protocols.",
			};

# GENERAL EXCEPTION CODES
		} elsif ($exceptionCode == 600) { # INTERNAL PROCESSING ERROR: [600] "A
generic catch-all for errors not specifically mentioned elsewhere in this list"

			($INB_Exception) = {
				'code' => $exceptionCode,
				'message' => "A generic error during internal processing.",
			};

		} elsif ($exceptionCode == 601) { # COMMUNICATION FAILURE: [601] "A
generic network failure"

			($INB_Exception) = {
				'code' => $exceptionCode,
				'message' => "A generic network failure.",
			};

		} elsif ($exceptionCode == 602) { # UNKNOWN STATE: [602] "Used when a
network call expects to find an existing state but failed"

			($INB_Exception) = {
				'code' => $exceptionCode,
				'message' => "Unknown State.",
			};

		} elsif ($exceptionCode == 603) { # NOT IMPLEMENTED: [603] "Not
implemented method in question"

			($INB_Exception) = {
				'code' => $exceptionCode,
				'message' => "Not implemented method in question.",
			};

# NUEVO-------------------------------------------------------
# NUEVO-------------------------------------------------------
		} elsif ($exceptionCode == 621) { # SERIVCE NOT AVAILABLE: [621] "Service
not available"

			($INB_Exception) = {
				'code' => $exceptionCode,
                        'message' => "Service not available.",
                };

# NUEVO-------------------------------------------------------
# NUEVO-------------------------------------------------------

# SERVICE INTRISIC ERRORS
        }  elsif  ($exceptionCode  ==  701)  {  #  SERVICE  INTERNAL  ERROR:  [701]
"Specific errors from the BioMOBY service"

                ($INB_Exception) = {
                        'code' => $exceptionCode,
                        'message' => "Specific errors from the BioMOBY service.",
                };

        } elsif ($exceptionCode == 702) { # OBJECT NOT FOUND: [702] "Object not
found with the given input"

                ($INB_Exception) = {
                        'code' => $exceptionCode,
                        'message' => "Object not found with the given input.",
                };

# NUEVO-------------------------------------------------------
# NUEVO-------------------------------------------------------

        } elsif ($exceptionCode == 721) { # INCORRECT ARTICLE NAME: [721] "The
specified name of MOBYData article is wrong or does not exist"

                ($INB_Exception) = {
                        'code' => $exceptionCode,
                        'message'  =>  "The  specified  name  of  MOBYData  article  is
wrong or does not exist.",
                };

        }  elsif  ($exceptionCode  ==  722)  {  #  INCORRECT  OBJECT  TYPE:  [722]
"Incorrect Object type from specified MOBYData article"

                ($INB_Exception) = {
                        'code' => $exceptionCode,
                        'message' => "Incorrect Object type from specified MOBYData
article.",
                };

        } elsif ($exceptionCode == 723) { # INCORRECT ARTICLENAME OBJECT: [723]
"The specified article name of BioMOBY Object is wrong or does not exist"

                ($INB_Exception) = {
                        'code' => $exceptionCode,
                        'message' => "The specified article name of BioMOBY Object
is wrong or does not exist.",
                };

        }  elsif  ($exceptionCode == 724)  {  #  INCORRECT  NAMESPACE  OBJECT:  [724]
"The namespace of specified BioMOBY Object is invalid"

                ($INB_Exception) = {
                        'code' => $exceptionCode,
                        'message' => "The namespace of specified BioMOBY Object is
invalid.",
                };

        } elsif ($exceptionCode == 731) { # INCORRECT ARTICLENAME OF SECONDARY:
[731] "The specified name of secondary is wrong or does not exist"

                ($INB_Exception) = {
                        'code' => $exceptionCode,
                        'message'  =>  "The  specified  name  of  secondary  is  wrong  or
does not exist.",
                };

        }  elsif  ($exceptionCode  ==  732)  {  #  INCORRECT  DATA  TYPE  OF  SECONDARY:
[732] "Incorrect data type from specified secondary article"

                ($INB_Exception) = {
                        'code' => $exceptionCode,
                              'message' => "Incorrect data type from specified secondary
article.",
                      };

               } elsif ($exceptionCode == 733) { # INCORRECT VALUE FROM SECONDARY: [733]
"The value of secondary article is invalid. It is not inside of correct range"

                      ($INB_Exception) = {
                              'code' => $exceptionCode,
                              'message' => "The value of secondary article is invalid. It
is not inside of correct range.",
                      };

               } elsif ($exceptionCode == 734) { # INCORRECT VALUE AND DEFAULT VALUE
FROM SECONDARY: [734] "There is not SECONDARY value and registered SECONDARY article has
not default value"

                      ($INB_Exception) = {
                              'code' => $exceptionCode,
                              'message' => "There is not SECONDARY value and registered
SECONDARY article has not default value.",
                      };
               }
# NUEVO------------------------------------------------------
# NUEVO------------------------------------------------------

} # End Switch

               return ($INB_Exception->{message});

} # End getExceptionCodeDescription

1;
```

- **testMobyException File:**

```
# #!/usr/local/bin/perl -w

# Perl pragma to restrict unsafe constructs
use strict;

# Issue warnings about suspicious programming.
use warnings;

use MobyException;


print "\n1. Test of exception reporting -------------------------------------------------
------------\n";
my ($excep);
eval {
		local(*INFILE);
		open(INFILE, "/usr/local/jm/toma") || die ($excep = MobyException->new(
									code => 200,
									queryID => 69,
									refElement => 'test',
							));
};
if ($@) {
		# Moby Exception
		if ($excep->isa('MobyException')) {
			print "Message: ".$excep->getExceptionMessage()."\n";
			print "Code: ".$excep->getExceptionCode()."\n";
			print "Error Response:\n".$excep-
>retrieveErrorExceptionResponse()."\n";
			print "Warning Response: \n".$excep-
>retrieveWarningExceptionResponse()."\n";
			print "Info Response:\n".$excep-
>retrieveInfoExceptionResponse()."\n";
			print "Empty MobyData Response:\n".$excep-
>retrieveEmptyMobyData()."\n";
			print "Empty MobySimple Response: \n".$excep-
>retrieveEmptyMobySimple()."\n";
			print "Empty MobyCollection Response:\n".$excep-
>retrieveEmptyMobyCollection()."\n";
		}
}


print "\n2. Test of exception reporting -------------------------------------------------
------------\n";
my ($excep2);
eval {
		local(*INFILE);
		open(INFILE, "/usr/local/jm/toma") || die ($excep2 = MobyException-
>new());
};
if ($@) {
		# Moby Exception
		if ($excep2->isa('MobyException')) {
			$excep2->setExceptionCode(201);
			$excep2->setExceptionMessage("Ahi estamos");
			$excep2->setExceptionMessage("Mejor este");

			print "Message: ".$excep2->getExceptionMessage()."\n";
			print "Code: ".$excep2->getExceptionCode()."\n";
			print "Error Response:\n".$excep2-
>retrieveErrorExceptionResponse()."\n";
			print "Warning Response: \n".$excep2-
>retrieveWarningExceptionResponse()."\n";
			print "Info Response:\n".$excep2-
>retrieveInfoExceptionResponse()."\n";
			print "Empty MobyData Response:\n".$excep2-
>retrieveEmptyMobyData()."\n";
			print "Empty MobySimple Response: \n".$excep2-
>retrieveEmptyMobySimple()."\n";
			print "Empty MobyCollection Response:\n".$excep2-
>retrieveEmptyMobyCollection()."\n";
		}
}
```

```
print "\n3. Test of exception reporting -------------------------------------------------
------------\n";
my ($test) = 'error';
if ($test eq 'error') {
              my ($excep3) = MobyException->new(queryID => 69, refElement => 'test');

              $excep3->setExceptionCode(221);
              $excep3->setExceptionMessage("Nombre del parametro");

              print "Message: ".$excep3->getExceptionMessage()."\n";
              print "Code: ".$excep3->getExceptionCode()."\n";
              print "Error Response:\n".$excep3->retrieveErrorExceptionResponse()."\n";
              print             "Warning             Response:             \n".$excep3-
>retrieveWarningExceptionResponse()."\n";
              print "Info Response:\n".$excep3->retrieveInfoExceptionResponse()."\n";
              print "Empty MobyData Response:\n".$excep3->retrieveEmptyMobyData()."\n";
              print       "Empty       MobySimple       Response:       \n".$excep3-
>retrieveEmptyMobySimple()."\n";
              print        "Empty        MobyCollection        Response:\n".$excep3-
>retrieveEmptyMobyCollection()."\n";
}
```