**Index**

Variables:

Methods:

Shortcut keys:

Transformer:

Filter:

Settings:

Example code:

**Variables**

|  |  |  |  |
| --- | --- | --- | --- |
| NAME | Symbol | scope | notes |
| globalmap | $g | All on mirth server |  |
| configurationmap | $cgf | All on mirth server | only get (put in settings tab) |
| globalChannelMap | $gc | Specified Channel (all messages) |  |
| channelmap | $c | Specified channel( for single message) |  |
| sourcemap | $s | Only on source(can defined on upstream process like preprocessor) | Read only |
| connectormap | $co | Only on that connector( destination1 not accessible on destinaton2) | In destionation |
| ResponseMap | $r | To Current message ,the destination responses stored automatically in this variable.(used in postprocessor/ source connector to send response back). | Readonly |
| lookupmap | $ | Picks ups variable based on all available variable maps. |  |

**GLOBAL script**   
only: global &configuration map variables are accessible

|  |  |  |  |
| --- | --- | --- | --- |
| Scripts | Execution sequence | execution | notes |
| deploy | - | When any channel deployed |  |
| undeploy | - | When any channel undeployed |  |
| preprocessor | Global preprocessor script then 🡪 channel preprocessor script | For each message is processed. This code executed. |  |
| Post processor | Channel postprocessor script🡪 global postprocessor script. | For each message processed |  |

**Mirth code notes.**

Note: globalMap, channelMap are same like Maps in javascript.  
 var globalMap=Map(); // initializing   
 globalMap.put(‘key1’,’value’); assigning values to map.  
 globalMap.get(‘key’); // retrieving values.

1. channelMap.put (‘key’ ,’value’) : stores the value with key in the channel level. This value is available as a channel variable with key name.

ex: channelMap.put (“name” , “sadik”);

1. channelMap.get(‘key’) : used to get the value of the variable stored with key name .

ex: var b=channelMap.get(‘name’) -🡪 key in quotation marks i.e $(‘key\_name’)

logger.info(b);

output: sadik.

1. globalMap.put (‘key’ ,’value’) : same as channelMap.put .but it can be accessed in outside

Of the channel.

1. globalMap.get(“key”): same as channelMap.get.

note: key should be in quotation marks (“ “). Quotation marks for value is necessary only if it is a string. if it’s a number not required. If passing variable, then also not required.

connectorMessage.getMessageId(): refers to the current message being processing which is received by source inbound (HL7) or generated with in channel. this is available in reference as messageId (connectorMessage.getMessageId()). If the message inbound is hl7 it returns the message controlled if it is a restful API, then it returns unique id generated by API server.  
  
 **REFERENCE methods/ maps**  
 1)channel  
 2)date  
 3)logging & alerts  
 4)mapping  
 5)database  
 6) message  
 7)fileReader   
 8)utility   
   
 1)channel :   
 1) channelName 🡪 var n1= channelName.toString();  
 2) channelId 🡪 var n2= channelId.toString();  
  
 2) date :  
 1) var dateString = DateUtil.getCurrentDate(pattern); 🡪 returns the current date in

in specified format.   
 2) var datestring = DateUtil.convertDate(inpattern, outpattern, date) 🡪 returns   
 passes date into given outpattern.  
 ex: var datestring = DateUtil.convertDate(“YYYYMMDD”, “YYYYDDMM”,”20210423”) 🡪 20242304  
 3) var dateString = DateUtil.formatDate(pattern, date);  
 4) var date = DateUtil.getDate(pattern, date);   
 3) logging& alerts  
 1) logger.info('message'); 🡪logs on info in serverlog(dashboard) 🡪 logger.info(“infor mation “);  
  
 2) logger.error('message') 🡪 same as logger.info🡪 used to log server error   
 3)alerts.sendAlert('message');   
  
4)mapping 🡪 map variable put & get functions.  
 ex: GlobalMap.put(“key1”,”value1”);

5)database: to connect to database   
 1) var dbConn;

var result;

try {

dbConn = DatabaseConnectionFactory.createDatabaseConnection('driver', 'address', 'username', 'password');

result = dbConn.executeUpdate('expression'); 🡪 “expression”= “SQL query”

} finally {

if (dbConn) {

dbConn.close();

}

}   
6)message:   
 1)iterate over all segment on message   
 2) iterate over a single segment   
 3)

DateUtil.convertDate(inpat tern, out pattern, date): this method is available in reference as convertdate. which is used to change the date pattern.   
ex: return DateUtil.convertDate('yyyyMMddHHmm', "yyyyMMdd'T'HHmm'00-0400'", date);

} or var date = return DateUtil.convertDate('yyyyMMddHHmm',

"yyyyMMdd'T'HHmm'00-0400'", date)

**TRANSFORMER**

Types:

1. Mapper: to create variables (channel Map, globalMap, connectorMap) in simple way without writing code. u can assingn values by drag and drop from message Tree or from reference (channelId etc).
2. javascript: to write the JavaScript code (majar code)

3) iterator:

4) message builder:

**FILTER**

Used to apply certain conditions on the code.

1)Rule Builder: uses if statement

Conditions:

1)exist: works on arrays only.

2)not exist: works on arrays only

3)equal: compares the field and values specified. u can drag and drop. Need to double click to enter the field / value area.

Ex: if(msg['MSH']['MSH.9']['MSH.9.2'].toString()=='A01’)

{

return true.

}

return false.

4) not equal: same as equal

5)contains: works on arrays only. Checks if the specified element is in array or not.

Give array name in field and element name in value area.

Ex: f ((myarray.indexOf(name) != -1)) {

return true.

}

return false.

6)not contains: same as contains.

2) JavaScript: to write user customized code.

3) iteration:

4)external script: not need now.

**Most commonly used codes:**

1. If, else if

Note: u can pass a variable as a condition in if block.

If the variable is not empty/not null == true

If the variable value is empty (‘’)/null ==false.

Ex: var name =” sadik”;

Var age=’’;

If(name)

{

console.log (“name defined”);

}

if(age){

console.log (“not defined”);

}

Output: name defined

1. For each: used to access the elements in an array.

Here to access the segments and fields in HL7 message.

Ex: var array=[1,2,3];

For each (x in array) // x is loop variable

{

Console.log(x);

}

Output: 1 2 3

Mirth ex:

1. Try catch used to handle the exceptions in code.

Ex: try {

Var b=mag [MSH].[MSH.5].toString();

If(!b) throw “ b is not defined/ is an empty string” ; // throw is like return in function

// code

}

catch(e) {

throw “exception in the code:”+e; // e is error message

}

finally {

logger.info(b); // code to be executed irrespective of exceptions in code

}

1. Array: most commonly used methods
2. . length : is the property.
3. .indexOf(element): returns index number of the element .
4. . push(element): used to add an element to the array.

ex

1. .Pop(): removes last element.
2. Function:
3. String: most commonly used methods.
4. .length : is the property .not a method in JavaScript.\*\*\*\*\*
5. .trim() : removes the white spaces in the beginning and at end of the string .
6. .split(‘ ‘), converts into array.
7. Object: stores in key-value pairs

Key and values can be any data types like string, number, array, function, Booleans or other objects.

1. Json: key must be in double quotation marks.

Ex: {

"name": "John",

"age": 30,

"isStudent": true,

"address": {

"street": "123 Main St",

"city": "New York"

}

}

Note: Values can only be of specific data types: string, number, Boolean, null, array, or

another JSON objects. Functions not allowed.

Method to covert javascript object to JSON: a= JSON.stringify(js\_objectName)

Arrays and functions: very important.

Array:

1. Declaring an array:

Var array=[ ]; // empty array \*\*\*\*

Var array1=[1,2,3,4,5,” sadik”,’pasha’]; // with elements

1. Inserting elements into array
2. Array [0] =” sadik”;
3. Array.push(element) \*\*\*\* // using push method passing element as argument .

Note: u can pass array as element by using the push method.

Ex: var a1=[1,2,3];

Var a2=[“sadik”];

a2.push(a1);

console.info(a2);

output: [“sadik”, [1,2,3]]

doubts:

segment? is it predefined or not ex: segment[‘MSH.2’][‘MSH.2.3’].toString();

for each loop:

notes: var xyz=msg; // stores entire HL7 message in xyz variable .

ex:

for each (segment in msg. children ()) // segment represents an element like x in for

each loop

{

var segment\_name = segment.name ();

if (segment\_name == 'NTE')

{

note\_parent. notes. push(convert\_note\_segment(segment));

}

}

Note: msg is the parent message

msg. children () is the child message

name() is the method on segment. It is mostly use in segment name comparison.

it returns segment name like MSH,EVN,PID.

If u print directly it returns: {

"uri": "",

"localName" : "IN1"

}

Note: use segment.name for comparison only. as given in the above code .

Get hl7 pdfs from komala.

**Irfan** (from KPI ONE NOTE)

**Upskill -**

HL7 2.8 certification

HIPAA Certification - In Progress

**Project Level -**

* Zero escalation.
* Complete the assigned EHR Interfaces.
* Understand data push - Need and How to do it.
* Improve skills on Mirth, Javascript and Rest API

MIRTH PDF notes

**LEARNINS**   
1)raw data – file – file (datatype- raw(all) onnectror- file reader,filewriter)  
2) hl7 to JSON - hl7-hl7-JSON (channel reared @ writer) -- code in destination transformer  
3) JSON to HL7 -- JSON -JSON-HL7 (“ “) -- code in destination transformer

4)JSON to CSV -- raw-raw-raw (“ ’) --“”

5) EDI X12 to JSON edi-edi-edi (“ “) --- “ “

6)sending msg from hl7 smart TCP sender (tcp sender : port) -- same

7)applying filters   
8) scripts

9) working with different source/destination connectors.

**CODE EXECUTION FLOW IN MIRTH**

1) SOURCE CONNECTOR : the inbound message is received

PRE-PROCESSORSCRIPT: preprocessor scrip executed. Used when needs to change the incoming messages. Ex: to select particular fields or to ignore some segments.

SOURCE FILTER: then the filter is execute( i.e particular inbound message is moved or not)

SOURCE TRANSFORMWER : transforms the message one format to other (hl7- JSON).

2)DESTINATION CONNECTOR :   
  
 1) DESTINATION FILTER: the messages from the source connector are after the source transformation are filtered here.  
 2) DESTINATION TRANSFORMER: the filtered messages are transformed (hl7 – JSON)   
3)POST PROCESSOR SCRIPT: executes after all the final transformation is done in the destination .  
4) DEPLOY SCRIPT: the deploy script is executed once (1) the channel is deployed. Only once.   
 ex: for the database connection.  
5) UNDEPLOY SCRIPT: this script is executed once after the channel is undeployed(only once ) ex: for the database disconnection before the channel is deleted.  
  
6) GLOBAL SCRIPT: the global script runs when the mirth connect is opened and this code is always available to all channel at all time.( used to access global variables , common functions).

For example, setting up a common database connection pool or defining utility functions.

The global script is loaded when Mirth starts and remains available throughout its runtime.

**DATABASE CONNECTOR:**

Use database writer connector :

\*\* 🡪 select the destination connector as :

🡪DATA BASE WRITER SETTINGS :

Driver🡪PostgreSQL 🡪 click on insert url template 🡪 change the (jdbc:postgresql://host:port /database)

Ex: jdbc:postgresql://localhost:5432/postgres.

-🡪 username : give username of the PostgreSQL(server) ex: postgres  
 password: password of PostgresSQL(server) ex: Sadik@06  
🡪 click on insert 🡪 choose the table 🡪 select the colums you want to insert into .  
🡪 click on generate   
🡪 it generates the SLQ query to insert data   
🡪 map the variables available on destination mapping box.  
 ex:

INSERT INTO patient\_data (name, id, DOB)

VALUES (, , ) 🡨 here drop and map.

Result: this values are inserted into the table.

var dbConn;

try {

dbConn=DatabaseConnectionFactory.createDatabaseConnection('org.postgresql.Driver','jdbc:postgresql://localhost:5432/postgres','postgres','Sadik@06');

} finally {

if (dbConn) {

dbConn.close();

}

}

**DOCUMENT WRITER CONECTOR**

DESTINATION:   
 **connector type:** document writer  
 **DOCUMENT WRITER SETTINGS**   
 **directory**: give the path where you want to create a pdf.

ex:C:/Users/SadikPasha/Documents/MIRTHPDF TEST   
 **filename** : give the name of the file ex: (patient\_data.pdf) 🡪 extension imp  
**HTML TEMPLATE:**  
 paste the template :  
 ex:

<html>

<head> <title> patient data</title></head>

<body>

<p> this is the pdf generated by mirth connect for the patient data </p>

<table border="2" height="50" width="50">

<tr ><td>NAME</td> <td>ID</td><TD>DOB</TD></tr>

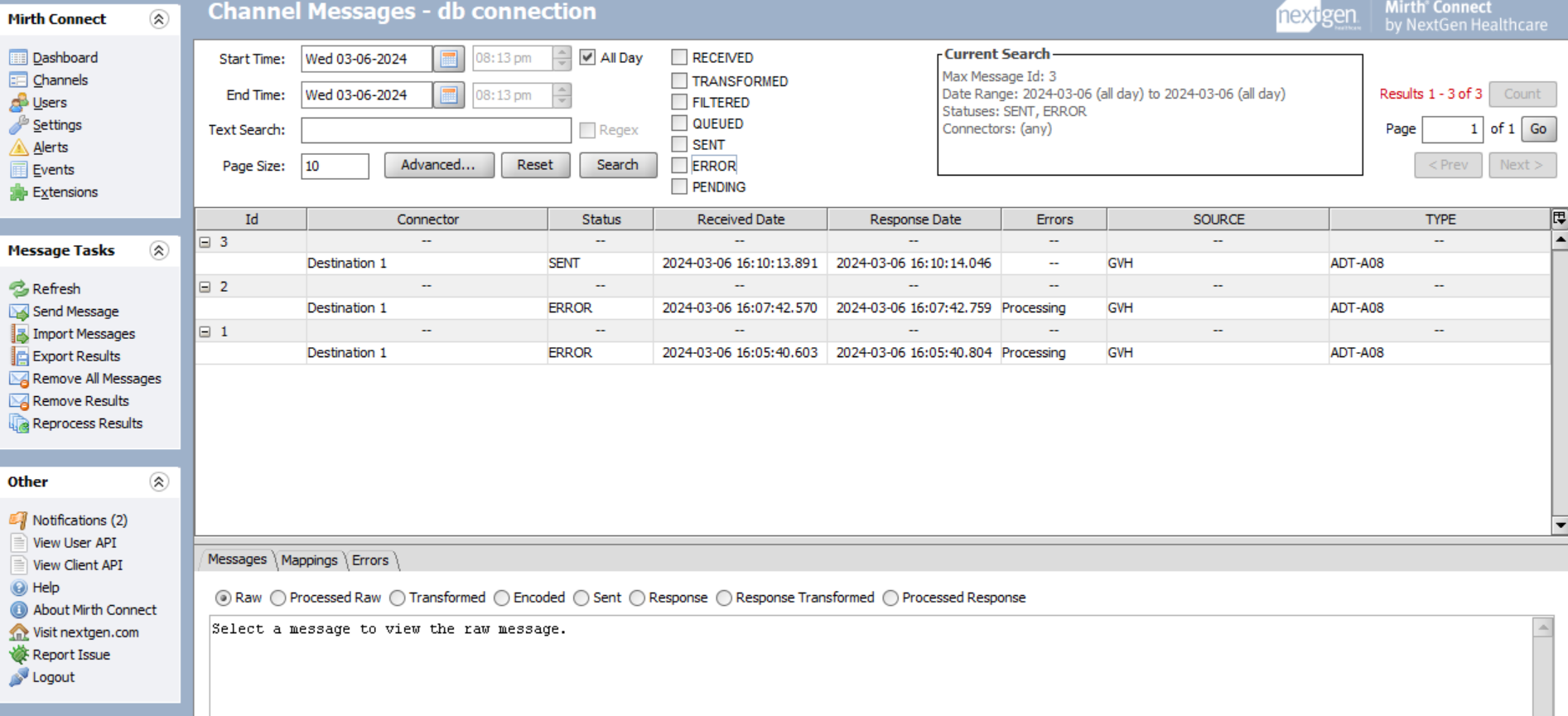
<tr><td>JON</td> <td>23</td><td>12-10-2000</td></tr>

</table>

</body>

</html>

Note : map the variables that you created in source connector in yellow color fields.(variable available in the destination mapping) ex: ${maps.get(‘name’)}   
  
result: the pdf file is generated on the provided location.

Dashboard TAB  
  


In the dashboard we can check the number of messages are processed through the channel .  
   
 1)by date : searching by date 2) by time on that date   
   
 start date ------ start time   
 end date-------- end time 0r (all day)

2)by message status 1) received  
 2)transformed  
 3) filtered   
 4)queued   
 5)sent   
 6) errored 🡪 click the check box

Note : then click search it shows all the messages.  
 if page size is 10 and messages are 10 or below the count button is not enabled.  
 it enabled only if the messages count is higher that the page size.  
 \*\*\* see results: 1:10 of 30 1🡪 page number   
 10🡪 number of messages in the page 1  
 30🡪 total number of messages.  
   
  
**INTERGATION & INTEROPERABILITY**

**INTERGATION**: Integration refers to the process of combining different components, systems, or entities to work harmoniously as a unified whole.  
 In the context of software and technology, integration typically involves connecting disparate systems, applications, or data sources to enable seamless communication, data sharing, and interoperability.   
1)data integration,   
2)application integration,   
3)system integration  
4) business process integration.   
  
note : integration(connection) is the process of connecting the different systems(amazon, phonepay, googlepay, mails, google location, mobile contacts data,multimedia(photo gallery access) to work as a single unit(amazon). Establishing a communication between the system. To interact with each other and to exchange the data.  
  
explanation: amazon aap is integrated with the phonepay . google pay payment gate ways.  
 amazon aap is integrated with the mobile contacts data for contact access.  
 amazon aap is integrated with the multimedia(photos, gallery ) to scam QR codes.  
  
   
 **INTEROERABILITY:**  
Interoperability refers to the ability of different systems, applications, or components to exchange a data seamlessly, effectively, and accurately. It ensures systems to work together regardless of their underlying technologies, platforms, or vendors. Interoperability enables data sharing, communication, and collaboration among disparate systems, thereby facilitating seamless interactions and integration across heterogeneous environments.  
  
notes: INTEROPERABILITY( inter+operation+ability): ability of systems to exchange data.  
 operation=communication.  
  
  
differences between integration and interoperability.  
 1) integration is connecting the systems for communication.(single time task)  
 2)interoperability is exchanging the data through integrated systems(continuous task)  
\*\*\* interoperability is achieved only if the systems are integrated(primary).  
  
   
**TRELLA HEALTH:**  
  
\*\* a healthcare company.  
 it collects data from the CMS(centers for Medicaid and Medicare).  
 1) collects data from providers EHR systems and Transforms to required formats using mirth.  
 2)send the data to the BPU a tool .here the data is processed according to the client requirements.  
 3) the processed data is shared to CRM tool.  
  
overview: some patients requires clinical assistance after discharging from the hospital ( post acute care).   
  
to get these services the doctors REFERRS the post acute service providers.  
  
trella take place here.  
  
trella provides a CRM tool that gives the details of the patients and provides requiring this services.  
  
the service providers sales team contact the patients using this tool.  
service types are:  
1) post acute care providers(multiple clients provides these service)  
2)HME (home medical equipment)  
3)infusion  
4)skilled Nursing  
  
 mirth: creates a separate interface in mirth (channels) for each clients.   
  
**FHIR to hl7**

\*\*\* similar as converting xml to HL7  
  
 as take source inbound messages as FHIR resource in XML format.  
 destination outnound as RAW(hl7).   
 map the data from the inbound fhir xml in destination transformer.

Ex: var patientId=msg['id']['@value'].toString();

var familyName=msg['name']['family']['@value'].toString();

var givenName=msg['name']['given']['@value'].toString();

var birthDate=msg['birthDate']['@value'].toString();

var gender=msg['gender']['@value'].toString();

var hl7Message =

'MSH|^~\&|||||||ADT^A08|||2.5\r' + // MSH segment

'PID|1||' + patientId + '||' + familyName + '^' + givenName + '||' + birthDate + '|' + gender + '||||' + // PID segment'||' + phoneNumber + '||||||||||||' + // PID segment continued

'|||||||||||' + '|||' + '\r'; // PID segment continued

msg=hl7Message;   
  
**XML/JSON ---HL7 v2.**   
  
note: it is very easy process of transformation.  
 SUMMARY: DATATYPE XML🡪 XML 🡪XML 🡪HL7  
 SOURCE 🡪 NO ANY CAHNGES  
 DESTINATION🡪   
 EDIT TRANSFORMAER :inbound message template (paste the incoming message)  
 : outbound message template(paste the HL7 message)  
  
 in java script: map the outbound hl7 message tree to the inbound xml message tree

ex: tmp['PID']['PID.5']['PID.5.1']=msg['name']['family']['@value'].toString();

tmp['PID']['PID.5']['PID.5.2']=msg['name']['given']['@value'].toString();   
  
 note : TEMPLATE Defines the structure of the message. If you use the hl7 template it maps aaccordingly.   
  
if mappings are there then those fields are mapped and changes occur in response.  
non-mapped hl7 outbound fields are set to the default as given in the template.  
   
 simply use the hl7 outbound message template to map the changes.  
  
 **USING TCP WRITER & TCP LISTENER.**

\*\* you can send/revoice message to the other system by using the IP address & port number.  
   
 1) channel1: source 🡪 channelReader , destination 🡪 TCP writer (default port 6660 )  
   
 2) channel2 : source🡪 TCP listener(6660) destination🡪 channelWtiter.  
   
 note: first deploy the channel2( as the TCP listener has to connect some system)  
   
 \*\* then deploy the channel1 and send a message.  
 \*\* the channel2 will listen the message from the channel1 and gives response.  
 note: for the remote system u have to change the host IP address as well.   
  
smart hl7 sender: just can be used as the remote tcp sender.  
 \*\* just create a channel and use the source connector as TCP listener(same port as given In

Smart hl7 sender) to receive messages. Destination --> your wish.

**CODE TEMPLATE :**  used to defines the libraries. Each library contains the code templates.  
 \*\* a code template is a method definition / drag -drop code block can be used on the channels.  
  
\*\* you can drag them as user defined functions like the built-in-methods from transformers under reference category.  
  
ex: /\*\*

Formats a given date with dashes ie. mm-dd-yyy

@param {String} date - any hl7 formatted date

@return {String} date formatted as mm-dd-yyyy

\*/

function dashDate(date) {

var yyyy = Number(date.toString().substring(0,4))

var mm = Number(date.toString().substring(4,6))

var dd = Number(date.toString().substring(6,8))

if (mm < 10) {

mm = '0' + mm

}

if (dd < 10) {

dd = '0' + dd

}

var dateReformatted = mm + '-' + dd + '-' + yyyy

return dateReformatted.toString()

}

function getCSV(csv) {

//load rows into csv.row array

for (var i = 0; i < msg.children().length(); i++) {

csv.row.push(msg['row'][i])

}

csv.row.sort()

}

**commonly used methods in mirth**

1) for each (seg in msg.children()) { // iterates over all segments in msg

if (seg.name().toString() == "SEG") {

var sample\_value = seg['SEG.1']['SEG.1.1'].toString();

}

}

2) for each (seg in msg..SEG) { // iterate over the particular segment

var sample\_value = seg['SEG.1']['SEG.1.1'].toString();

}note:in methods lowercase are default(fixed). Only upper changes required. Ex : seg is default , SEG needs to change as original I.e. SEG🡪 MSH/PID/NK1.