Steps :  
  
1) create a new channel: summery : datatype: rawa-raw-raw-raw.  
 source: javascript reader :  
 poll once on : set time ( 08:30pm)  
 paste the fallowing code :

Mirth Username =sadikpasha,   
 password=sadik@06mca  
 server=127.0.0.1  
 backup destination path='C:\\Users\\SadikPasha\\Documents\\mirthAutobackupfolde.  
🡪 deploy the channel.  
---------------------------------------------------------------------------------------------------

backupMirthServer('sadikpasha','Sadik@06mca','127.0.0.1','C:\\Users\\SadikPasha\\Documents\\mirthAutobackupfolder');

/\*\*

Creates a backup of the whole mirth configuration (like if you press the "backup config" button in Mirth Administrator)

and writes it to a folder.

The name of each backup is unique & all backups of one day will be written to a dedicated sub-folder named by the date.

@param {String} username - The username that the channel should use to connect to the server that should be backed-up.

@param {String} password - The password that the channel should use to connect to the server that should be backed-up.

@param {String} server - The ip or name of the mirth server that should be backed-up. This parameter will become part of the backup name.

@param {String} backupFolder - Path to the folder where the backup should be created

@param {String} archivePassword - If a password is provided, the zip archive will be encrypted with this password

\*/

function backupMirthServer(username, password, server, backupFolder, archivePassword) {

logger.info('Initializing export of mirth server ' + server);

// create a client instance and initialize it with the server to which it should connect

var client = new com.mirth.connect.client.core.Client('https://' + server + ':8443');

// create an instance of the serializer used to serialize the configuration to xml

var serializer = com.mirth.connect.model.converters.ObjectXMLSerializer.getInstance();

// log on to the server

try{

var loginStatus = client.login(username, password);

}catch(ex){

throw 'Unable to log-on the server "' + server + '" with credentials ' + username + '/' + password + ' (incompatible mirth version?): ' + ex.message;

}

// check if login was successful

if (loginStatus.getStatus() != com.mirth.connect.model.LoginStatus.Status.SUCCESS) {

logger.error('Unable to log-on the server "' + server + '" with credentials ' + username + '/' + password + '(status ' + loginStatus.getStatus() + ')');

return;

}

try {

// get the server configuration

var configuration = client.getServerConfiguration();

//get the code template libraries

var codeTemplateLibrary = client.getServerConfiguration().getCodeTemplateLibraries();

// get the current date as string

var backupDate = new String(DateUtil.getCurrentDate('yyyy-MM-dd HH:mm:ss'));

var todaysFolder = new String(DateUtil.getCurrentDate('yyyy-MM-dd'));

// generate the complete backupPath of the backup file

var backupFolder = (new String(backupFolder)).replace(/\\/g, '/') + '/' + todaysFolder;

// set the date of the backup in the server configuration

configuration.setDate(backupDate);

// create the directory if not existant

org.apache.commons.io.FileUtils.forceMkdir(new java.io.File(backupFolder));

// 1.) export the mirth configuration to the archive

// create an xml representation of the configuration object

configuration = serializer.serialize(configuration);

// create a streem from the xml

logger.info('Exporting configuration of mirth server "' + server + '"');

var file1=backupFolder + "/" + todaysFolder + " Mirth Non-Prod Backup.xml";

// and write the file to the archive

FileUtil.write(file1, false, configuration);

//2.) export the configuration map to the archive

// var configMap = getConfigurationProperties(client);

// var file2=backupFolder + "/" + "Configuration Map.properties";

// and write the file to the archive

// FileUtil.write(file2, false, configMap);

//3.) export all channels

// var channels = getChannels(client, backupFolder, serializer);

//4.) export all code template libraries

// var codeTemplates = getCodeTemplates(client, backupFolder, serializer, codeTemplateLibrary);

// end the session

client.logout();

// and close the client instance

//client.close();

logger.info('Configuration of mirth server "' + server + '" has been exported to "' + backupFolder + '"');

} catch (ex) {

logger.error('unable to write file "' + backupFolder + '": ' + ex.message);

} finally{

try{file.close();}catch(e){}

try{configMap.close();}catch(e){}

}

}

/\*\*

Provides the configuration map as an imput stream.

@param {Object} client - The mirth client instance for the server of which the configuration map should be exported.

@return {InputStream} The confguration map in the format like in the <b><i>configuration.properteis</b></i> file

\*/

function getConfigurationProperties(client){

// prepare structure

var properties = new org.apache.commons.configuration.PropertiesConfiguration();

// no fancy parsing here just a standard container

properties.setDelimiterParsingDisabled(true);

properties.setListDelimiter(0);

properties.clear();

var layout = properties.getLayout();

// order the properties - basically just like the mirth admin does

var sortedMap = java.util.TreeMap(java.lang.String.CASE\_INSENSITIVE\_ORDER);

sortedMap.putAll(client.getConfigurationMap());

// change the layout for obtainin

for (var iterator = sortedMap.entrySet().iterator(); iterator.hasNext();) {

var entry = iterator.next();

var key = entry.getKey();

// if the key is left emty, this entry is invalid and therefore skipped

if(!key){continue;}

var value = entry.getValue().getValue();

var comment = entry.getValue().getComment();

properties.setProperty(key, value);

layout.setComment(key, comment ? comment : null);

}

// now write the file to a stream. Let's do everything on the fly

var exportMap = new java.io.ByteArrayOutputStream();

properties.save(exportMap);

// provide an input stream that can directly be written to the archive

return exportMap;

}

/\*\*

Provides the configuration map as an imput stream.

@param {Object} client - The mirth client instance for the server of which the configuration map should be exported.

@param {String} backupFolder - Path to the folder where the backup should be created.

@param {Object} backupFolder - the serializer used to serialize the configuration to xml

\*/

function getChannels(client, backupFolder, serializer){

var sortedSet = java.util.TreeSet(java.lang.String.CASE\_INSENSITIVE\_ORDER);

sortedSet.addAll(new com.mirth.connect.server.userutil.ChannelUtil.getChannelIds());

var channels = client.getChannels(sortedSet);

for(var i = 0; i<channels.size(); i++){

channelData = serializer.serialize(channels.get(i));

var file=backupFolder + "/Channels/" + channels.get(i).getName() + ".xml";

// and write the file to the archive

FileUtil.write(file, false, channelData);

}

return;

}

function getCodeTemplates(client, backupFolder, serializer, codeTemplateLibrary){

var codeTemplateLibraryIds = java.util.TreeSet(java.lang.String.CASE\_INSENSITIVE\_ORDER);

for(var j=0; j< codeTemplateLibrary.size(); j++){

codeTemplateLibraryIds.add(codeTemplateLibrary.get(j).getId());

}

var codeTemplates = client.getCodeTemplateLibraries(codeTemplateLibraryIds, true);

codeTemplate = serializer.serialize(codeTemplates);

var file=backupFolder + "/" + "Code Templates Backup.xml";

// and write the file to the archive

FileUtil.write(file, false, codeTemplate);

}