TigaseDoc

Release 0.1

Tigase, Inc.

Jun 21, 2023

CONTENTS

Welcome to Tigase PubSub component guide

CHAPTER

ONE

PUBSUB COMPONENT

Tigase's Publish Subscribe component is an XEP-0060 compliant plugin handling all publish and subscribe activity within Tigase server. This is enabled as default with the pubsub name, however you may include the following line if you wish to customize it's configuration.

```
pubsub () {}
```

You may change the name so long as you specify the pubsub class within parenthesis.

1.1 Tigase Pubsub Release Notes

Welcome to Tigase Pubsub 5.0.0! This is a feature release for with a number of fixes and updates.

1.1.1 Tigase PubSub 5.1.0 Release Notes

All Changes

- Added support for mam2#extended; #mam-73
- Improving processing of presence and sending last published item; #pubsub-133
- Fix NPE in UnsubscribeNodeModule; #pubsub-130
- Fixed issue with MAM returning incorrect entries for MAM capable PubSub nodes; #pubsub-131

1.1.2 Previous Releases

1.1.3 Tigase PubSub 5.0.0 Release Notes

Major Changes

- Add publishing executor with rate limiting
- Optimisations and fixes

All Changes

- #pubsub-102: Add publishing executor with rate limiting
- #pubsub-103: Empty message notification id attribute
- #pubsub-105: NPE in RetrieveItemsModule
- #pubsub-106: NPE in PubsubPublishModule?Eventbus
- #pubsub-107: disco#items feature returned on disco#info request for PubSub node item
- #pubsub-108: Fix Missing notification for published events
- #pubsub-110: Fix Deadlock in TigPubSubRemoveService SP on MySQL
- #pubsub-111: Fix SQLException: At least one parameter to the current statement is uninitialized.
- #pubsub-113: Fix StackOverflowError in LRUCacheWithFuture
- #pubsub-114: Fix pubsub#persist_items is not advertised
- #pubsub-115: Fix Cannot add or update a child row: a foreign key constraint fails (`tigasedb.tig_pubsub_items`, CONSTRAINT `tig_pubsub_items_ibfk_1` FOREIGN KEY (`node_id`) REFERENCES `tig_pubsub_nodes` (`node_id`))
- #pubsub-119: Fix NPE in DiscoveryModule
- #pubsub-120: Fix Empty element in POST payload is incorrectly parsed
- #pubsub-121: Use String.intern() for PEP CAPS nodes string
- #pubsub-124: Fix PubSub sends notifications about last published item on each presence received from subscriber.
- #pubsub-125: Reported features `pubsub#metadata` should be `pubsub#meta-data`
- #pubsub-126: Fix Deadlocks in MySQL schema
- #pubsub-127: Fix NPE in UserEntry.remove
- #pubsub-128: Fix PatternSyntaxException for users with emoticons in resource

Announcement

Major changes

Tigase pubsub component has undergone a few major changes to our code and structure. To continue to use Tigase pubsub component, a few changes may be needed to be made to your systems. Please see them below:

Database schema changes

Current version comes with changes to database schema to improve JID comparison during lookup of nodes, subscriptions, affiliations, etc.

To continue usage of new versions of pubsub component it is required to manually load new component database schema, see *database preparation* section for more information.

> **Warning:** Loading of new database schema is required to use new version of pubsub component.

Changes in REST API

We continuously work on improving usability and making our REST API easier to use we added support for handling JSON requests in REST API for pubsub. At the same time we decided to slightly modify responses in XML sent by REST API to make responses in JSON and XML similar.

For more informations about current REST API please look into *Rest API* section.

New features

Support for using separate database for different domains

Since this version it is possible to use separate pubsub nodes and items based on domains. This allows you to configure component to store informations about nodes and items for particular domain to different database.

For more informations please look into *using multiple databases*.

Support for MAM

In this version we added support for XEP-0313: Message Archive Management protocol which allows any MAM compatible XMPP client with pubsub support to retrieve items published on pubsub nodes using MAM protocol for querying.

1.2 Configuration

1.2.1 Pubsub naming

Within Tigase, all pubsub component address MUST be domain-based address and not a JID style address. This was made to simplify communications structure. Tigase will automatically set component names to pubsub.domain, however any messages send to pubsub@domain will result in a `SERVICE_UNAVAILABLE` error.

Pubsub nodes within Tigase can be found as a combination of JID and node where nodes will be identified akin to service discovery. For example, to address a friendly node, use the following structure:

```
<iq to='pubsub.domain'>
  <query node='friendly node'/>
</iq>
```

1.2.2 Configure Roster Maximum size

Administrators can configure the maximum allowable roster size per user via the config.tdsl file.

```
'sess-man' {
    'jabber:iqa:roster' {
        max_roster_size = '100'
    }
}
```

This sets the roster limit to 100 entries per user. It can be set to any integer, however by default no limit is set and no configuration is set in `config.tdsl` file.

1.2.3 Store Full XML of Last Presence

Tigase can store a more detailed <unavailable/> presence stanza to include timestamps and other information.

Requirements

Ensure that `presence-offline` plugin is enabled in config.tdsl. To do this, add be sure `presence-offline` is listed under `sess-man`

```
'sess-man' {
    'presence-offline' () {}
}
```

The following two lines in `sess-man` configure options to broadcast probes to offline users.

```
'sess-man' {
    'skip-offline' = 'false'
    'skip-offline-sys' = 'false'
}
```

Without these lines, Tigase will not send presence probes to users that the server knows to be offline.

The full XML presence is stored under the tig_pairs table with a pkey of `last-unavailable-presence` will look like this:

```
<presence from="user@example.com" xmlns="jabber:client" type="unavailable">
<status>Logged out</status>
<delay stamp="2015-12-29T16:51:50.748Z" xmlns="urn:xmpp:delay"/></presence>
```

As you can see, the plugin has added a delay stamp which indicates the last time they were seen online. This may be suppressed by using the following line in your config.tdsl file.

```
'sess-man' {
    'delay-stamp' = 'false'
}
```

You may also limit probe responses only to newly connected resources.

```
'sess-man' {
    'probe-full-jid' = 'true'
}
```

When a user logs on, they will receive the same full unavailable presence statements from contacts not logged in. Also the repository entry containing their last unavailable presence will be removed.

NOTE: This will increase traffic with users with many people on their rosters.

1.2.4 Using separate store

As mentioned above, by default Tigase pubsub component uses default data source configured for Tigase XMPP Server. It is possible to use separate store by pubsub component. To do so you need to configure new `DataSource` in `dataSource` section. Here we will use `pubsub-store` as name of newly configured data source. Additionally you need to pass name of newly configured data source to `dataSourceName` property of default DAO of pubsub component.

```
dataSource {
    pubsub-store () {
        uri = 'jdbc:postgresql://server/pubsub-database'
    }
}

pubsub () {
    dao {
        default () {
            dataSourceName = 'pubsub-store'
        }
    }
}
```

It is also possible to configure separate store for particular domain, ie. `pubsub.example.com`. Here we will configure data source with name `pubsub.example.com` and use it to store data for pubsub nodes and items at `pubsub.example.com`:

```
dataSource {
    'pubsub.example.com' () {
        uri = 'jdbc:postgresql://server/example-database'
    }
}

pubsub () {
    dao {
        'pubsub.example.com' () {
          # we may not set dataSourceName as it matches name of domain
        }
    }
}
```

Note: With this configuration, data for other domains than `pubsub.example.com` will be stored in default data source.

1.2.5 Enabling PEP support

To enable XEP-0163: Personal Eventing Protocol support it is required to set `persistent-pep` property of pubsub component to `true`, set `send-last-published-item-on-presence` property of component to `true` and enable `pep` SessionManager processor.

```
pubsub () {
    persistent-pep = true
    send-last-published-item-on-presence = true
}

sess-man () {
    pep () {
    }
}
```

Note: If your pubsub component uses different name than `pubsub` then you need to set `pubsub-jid` property of `pep` processor to JID of pubsub component make it aware of a different name of a pubsub component.

Example with pubsub component named ``events`` hosted at server named ``servername.com`` and enabled PEP.

```
events () {
    persistent-pep = true
    send-last-published-item-on-presence = true
}
sess-man () {
    pep () {
        'pubsub-jid' = 'events@servername.com'
    }
}
```

1.2.6 Enabling REST API

To use REST API for pubsub component it is required that:

- Tigase HTTP API component is installed and configured properly. For information about HTTP API component installation please look into *HTTP component documentation*.
- Tigase pubsub REST scripts are copied to HTTP API REST scripts directory In installation package this is already done and scripts are in proper locations. dd* JID of HTTP API component needs to be added to list of trusted jids of Tigase pubsub component `trusted` property (if `http` is name of HTTP API component)

```
pubsub () {
    trusted = [ 'http@{clusterNode}' ];
}
```

1.2.7 Changing nodes cache size

By default Tigase pubsub component caches node configuration of 2000 last loaded nodes. If there are many requests to database to load node configuration and your installation contains many nodes it may be a good idea to increase number of cached nodes.

To do this you need to set `pubsub-repository-cache-size` property of pubsub component to new size.

```
pubsub () {
    pubsub-repository-cache-size = 4000
}
```

Enable sending last published item on presence

By default it is not possible to use delivery of last published item when users broadcasts initial presence. To do so you need to set `send-last-published-item-on-presence` of pubsub component to `true`. This will allow you to configure nodes to send last published item on presence.

```
pubsub () {
    send-last-published-item-on-presence = true
}
```

1.2.8 Throttling sending notifications

Notifications sent by PubSub component may be sent in large batches, if you have a nodes with a lot of subscribers. In those cases, it is useful to throttle publications to improve behaviour and performance of other parts of Tigase XMPP Server.

To achieve that, PubSub throttles generate notifications to specified throughput. By default it is set to 5k packets for each CPU core available per second.

To set it to a different value, you can set `limit` property of `publishExecutor` bean to the expected number of publications per second, ie. 100000;

Note: This value is a number of total throughput, and will not be adjusted by the number of available CPU cores.

```
pubsub () {
    publishExecutor () {
        limit = 10000
    }
}
```

Publication rate is also adjusted to current memory usage on a 4 point scale adjusted to the value of two configuration options: `highMemoryUsageLimit` and `criticalMemoryUsageLimit` (with default values: 90% and 98% respectively): * `normal` - if memory usage is below `highMemoryUsageLimit` (i.e. below 90%) * `high` - memory usage less than halfway between `highMemoryUsageLimit` and `veryHigh` (i.e. between 90% and 94%) * `veryHigh` - memory usage more than halfway between `highMemoryUsageLimit` and `veryHigh` (i.e. between 94% and 98%) * `critical` - if memory usage is above `criticalMemoryUsageLimit` (i.e. above 98%)

It's possible to adjust values of the high and critical limits in publisher bean:

```
pubsub () {
    publishExecutor () {
        highMemoryUsageLimit = 90
        criticalMemoryUsageLimit = 98
    }
}
```

1.2.9 Disable automatic subscription of node creator

During creation of node pubsub component subscribes creator to pubsub node and delivers notifications to creator. If in your case you do not want this behavior, you may set `auto-subscribe-node-creator` property of pubsub component to `false`.

```
pubsub () {
    auto-subscribe-node-creator = false
}
```

1.3 Database

1.3.1 Preparation of database

Before you will be able to use Tigase PubSub Component you need to initialize database. We provide few schemas for this component for MySQL, PostgreSQL, SQLServer and DerbyDB.

They are placed in `database/` directory of installation package and named in `dbtype-pubsub-version.sql`, where `dbname` in name of database type which this schema supports and `version` is version of a PubSub component for which this schema is designed.

You need to manually select schema for correct database and component and load this schema to database. For more information about loading database schema look into *database preperation* section of this guide.

1.3.2 Upgrade of database schema

Database schema for our components may change between versions and if so it needs to be updated before new version may be started. To upgrade schema please follow instructions from the *database preperation* section.

Note: If you use SNAPSHOT builds then schema may change for same version as this are versions we are still working on.

1.3.3 Schema description

Tigase PubSub component uses few tables and stored procedures. To make it easier to identify tables and stored procedures used by PubSub component they are prefixed with `tig_pubsub_`.

Table `tig_pubsub_service_jids`

This table stores all jids for which PubSub component contains nodes.

Field	Description	Comments
service_id	Database ID of a service JID	
service_jid	Value of a service JID	
service_jid_sha1	SHA1 value of lowercased service JID	Used for proper bare JID comparison during lookup. (N/A to PostgreSQL schema)

Table `tig_pubsub_jids`

This table stores all jids related to PubSub nodes, ie. subscriber, affiliates, creators, publishers, etc.

Field	Description	Comments
jid_id	Database ID of a bare JID	
jid	Value of a bare JID	
jid_sha1	SHA1 value of lowercased bare JID	Used for proper bare JID comparison during lookup. (N/A to PostgreSQL schema)

Table `tig_pubsub_nodes`

Table stores nodes tree structure and node configuration.

Field	Description	Comments
node_id	Database ID of a node	
service_id	ID of service JID	References `service_id` from `tig_pubsub_service_jids`
name	Name of PubSub node	
name_sha1	SHA1 of PubSub node name	Used for indexing and faster lookup. (N/A to PostgreSQL schema)
type	Type of PubSub node	0 - collection 1 - leaf
title	Title of PubSub node	
description	Description of a node	
creator_id	ID of JID of creator	References `jid_id` from `tig_pubsub_jids`
creation_date	Timestamp of creation time	
configuration	Serialized configuration of PubSub node	
collection_id	Points collection (parent node)	References `node_id` from `tig_pubsub_nodes`

Table `tig_pubsub_affiliations`

Table stores affiliations between PubSub nodes and jids.

Field	Description	Comments
node_id	ID of a node	References `node_id` from `tig_pubsub_nodes`
jid_id	ID of a user JID	References `jid_id` from `tig_pubsub_jids`
affiliation	Affiliation value	

Table `tig_pubsub_subscriptions`

Table stores subscriptions of jids to PubSub nodes.

Field	Description	Comments
node_id	ID of a node	References `node_id` from `tig_pubsub_nodes`
jid_id	ID of a user JID	References `jid_id` from `tig_pubsub_jids`
subscription	Subscription value	
subscription_id	Id of a subscription	

Table `tig_pubsub_items`

Table stores items of PubSub nodes.

Field	Description	Comments
node_id	ID of a node	References `node_id` from `tig_pubsub_nodes`
id	Id of an items	
id_sha1	SHA1 of item id	Indexed and used for faster lookup (N/A to PostgreSQL schema)
creation_date	Creation date	
publisher_id	ID of publisher JID	References `jid_id` from `tig_pubsub_jids`
update_date	Timestamp of last item modification	
data	Item payload	

1.3.4 PubSub Schema Changes

Tigase PubSub Component is currently version 3.3.0 which is introduced in Tigase server v8.0.0.

PubSub 3.2.0 Changes

PubSub v 3.2.0 adds a new procedure TigPubSubGetNodeMeta which supports PubSub metadata retrieval while conducting a disco#info query on nodes.

You will need to upgrade your database if you are not using v3.2.0 schema. Tigase will report being unable to load PubSub component if you do not have this schema version.

The MySQL schema can be found Here.

The Derby schema can be found Here.

The PostGRESQL schema can be found Here.

The MS SQL schema can be found Here.

The same files are also included in all distributions of v8.0.0 in `[tigaseroot]/database/` . All changes to database schema are meant to be backward compatible.

For instructions how to manually upgrade the databases, please refer to *Tigase v7.1.0 Schema Updates section*.

Upgrading older installations (pre-v3.0.0 Schema)

To update older installations of Tigase to the PubSub Schema v3.0.0 follow these instructions. Note this should be done before upgrading to PubSub v3.1.0.

Step by Step guide.

Prepare Old Database for Upgrade

In `database` directory of Tigase installation you will find SQL files which will prepare old database schema for upgrade using following this naming pattern: `<database_type>-pubsub-schema-3.0.0-pre-upgrade.sql` Where `<database_type>` can be one of the following: `mysql`, `sqlserver`, ie. for MySQL you will find the file `mysql-pubsub-schema-3.0.0-pre-upgrade.sql`. You need to execute statements from this file on your source database, which will drop old procedures and functions used to access database and also this statements will rename old tables by adding suffix _1 to each of old tables. Example:

MySQL

```
mysql -u tigase -p tigase_pubsub < database/mysql-pubsub-schema-3.0.0-pre-upgrade.
sql
```

MS SQL

```
sqlcmd -S %servername% -U %root_user% -P %root_pass% -d %database% -i database\
sqlserver-pubsub-schema-3.0.0-pre-upgrade.sql
```

Update Tigase PubSub Component

For this you need to copy the Tigase PubSub Component jar file to jars directory inside Tigase XMPP Server installation directory. It is also recommended to copy files from database directory of Tigase PubSub Component to database directory in Tigase XMPP Server installation directory.

If you happen to use one of the the distribution packaged (either installer or -dist-max flavored archive) then all required files are already available - both new schema files will be available in `database/` directory as well as both versions of PubSub component will be present in `jars/` directory - PubSub3 as tigase-pubsub.jar and PubSub2 as tigase-pubsub-2.2.0.jar.old (provided for compatibility reasons).

Load New Schema

In the `database` directory you will find files containing new schemas for:

- MySQL - `mysql-pubsub-schema-3.0.0.sql`
- PostgreSQL - `postgresql-pubsub-schema-3.0.0.sql`
- MSSQL - `sqlserver-pubsub-schema-3.0.0.sql`
- DerbyDB - `derby-pubsub-schema-3.0.0.sql` and `pubsub-db-create-derby.sh`

For most databases, with the exception of Derby, you only need to execute statements from the proper file. For example:

MySQL

```
mysql -u tigase -p tigase_pubsub < database/mysql-pubsub-schema-3.0.0.sql
```

MS SQL

```
sqlcmd -S %servername% -U %root_user% -P %root_pass% -d %database% -i database\
sqlserver-pubsub-schema-3.0.0.sql
```

PostgreSQL

```
psql -h $DB_HOST -q -U ${USR_NAME} -d $DB_NAME -f database/sqlserver-pubsub-schema-3.
0.0.sql
```

For DerbyDB you need to execute the `pubsub-db-create-derby.sh` script and pass proper JDBC URI to database to which you want to load schema (if database does not exist, it will be created).

```
database/pubsub-db-create-derby.sh
```

NOTE: It is possible to use same database which was used before - then after upgrade you will have new tables and old tables with _1 suffix.

Execute Migration Utility

In the `/database` directory you will find the `pubsub-db-migrate.sh` file which you need to execute and pass arguments with JDBC URIs needed to connect to source and destination database. If you used dedicated tables for PubSub you will also need to pass a class name used to access database (value of `pubsub-repo-class` variable from `etc/config.tdsl` file).

Example for dedicated table used for PubSub:

```
database/pubsub-db-migrate.sh -in-repo-class tigase.pubsub.repository.PubSubDAO
-in 'jdbc:mysql://localhost/tigase_pubsub?user=tigase&password=passwd'
-out 'jdbc:mysql://localhost/tigase_pubsub?user=tigase&password=passwd'
```

Example for use without dedicated PubSub tables:

```
database/pubsub-db-migrate.sh
-in 'jdbc:mysql://localhost/tigase?user=tigase&password=passwd'
-out 'jdbc:mysql://localhost/tigase?user=tigase&password=passwd'
```

Example for use with dedicated tables in a Windows environment:

```
database/pubsub-db-migrate.cmd -in-repo-class tigase.pubsub.repository.PubSubDAO
-in 'jdbc:sqlserver://<hostname>\\<instance>:<port>;databaseName=<name>;user=tigase;
↪password=tigase;schema=dbo;lastUpdateCount=false'
-out 'jdbc:sqlserver://<hostname>\\<instance>:<port>;databaseName=<name>;user=tigase;
↪password=tigase;schema=dbo;lastUpdateCount=false'
```

During execution this utility will report information about migration of PubSub data to the new schema, and the same information will be store in `pubsub_db_migration.log`.

Finish

After successful migration you will have all data copied to new tables. Old tables will be renamed by adding suffix _1. After verification that everything works OK, you can delete old tables and it's content as it want be used any more.

1.4 Features

1.5 AdHoc Commands

Similar to the HTTP API, AdHoc commands based on groovy scripts can be sent to this component to do a number of tasks. All scripts for these Ad-hoc commands are found at `sec/main/groovy/tigase/admin` in source distributions, or at this link. To use them, the scripts need to be copied into the `scripts/admin/pubsub` folder in the Tigase installation directory. For all examples, the component address will be `pubsub.example.com`.

1.5.1 Create a Node

Ad-hoc command node: `create-node` Required role: Service Administrator

Command requires fields `node` and `pubsub#node_type` to be filled with proper values for execution. - `node` Field containing id of node to create. - `pubsub#node_type` Contains one of two possible values. * `leaf-node` Node that will be published. * `collection` Node that will contain other nodes.

Other fields are optional fields that can be set to change configuration of newly create node to different configuration than default.

Example call using TCLMT:

```
bin/tclmt.sh -u admin@example.com -p admin123 remote pubsub.example.com create-node␣
↪example admin@example.com leaf
```

1.5.2 Delete a Node

Ad-hoc command node: `delete-node` Required role: Service Administrator

Command requires `node` field to be filled. - `node` Field containing id of node to delete.

Example call using TCLMT:

```
bin/tclmt.sh -u admin@example.com -p admin123 remote pubsub.example.com delete-node␣
↪example
```

1.5.3 Subscribe to a Node

Ad-hoc command node: `subscribe-node` Required role: Service Administrator

Command requires `node` and `jids` nodes to be filled. - `node` Field containing node to subscribe to. - `jids` Field containing list of JIDs to subscribe to the node.

Example call using TCLMT:

```
bin/tclmt.sh -u admin@example.com -p admin123 remote pubsub.example.com subscribe-node␣
↪example admin@example.com,test1@example.com
```

1.5.4 Unsubscribe to a Node

Ad-hoc command node: `unsubscribe-node` Required role: Service Administrator

Command requires `node` and `jids` nodes to be filled. - `node` Field containing node to unsubscribe to. - `jids` Field containing list of JIDs to unsubscribe to the node.

Example call using TCLMT:

```
bin/tclmt.sh -u admin@example.com -p admin123 remote pubsub.example.com unsubscribe-node␣
↪example admin@example.com,test2@example.com
```

1.5.5 Publish an item to a Node

Ad-hoc command node: `publish-item` Required role: Service Administrator

Command requires fields `node` and `entry` to be filled. - `node` Field containing id of node to publish to. - `item-id` Field may contain id of entry to publish, can be empty. - `entry` Field should contain multi-line entry content that should be valid XML values for items.

This command due to it's complexity cannot be easily executed by TCLMT using default remote script which provides support for basic adhoc commands. Example call using TCLMT:

```
bin/tclmt.sh -u admin@example.com -p admin123 remote pubsub.example.com publish-item␣
↪example item-1 '<entry><title>Example 1</title></entry>'
```

Example Groovy script to execute create-node command using JAXMPP2

```
import tigase.jaxmpp.j2se.Jaxmpp
import tigase.jaxmpp.core.client.AsyncCallback
import tigase.jaxmpp.core.client.exceptions.JaxmppException
import tigase.jaxmpp.core.client.xmpp.stanzas.Stanza
import tigase.jaxmpp.core.client.SessionObject
import tigase.jaxmpp.j2se.ConnectionConfiguration
import tigase.jaxmpp.core.client.xml.Element
import tigase.jaxmpp.core.client.xml.DefaultElement
import tigase.jaxmpp.core.client.xmpp.forms.JabberDataElement

Jaxmpp jaxmpp = new Jaxmpp();

jaxmpp.with {
    getConnectionConfiguration().setConnectionType(ConnectionConfiguration.
↪ConnectionType.socket)
    getConnectionConfiguration().setUserJID("admin@example.com")
    getConnectionConfiguration().setUserPassword("admin123")
}

jaxmpp.login(true);
```

(continues on next page)

(continued from previous page)

```
def packet = IQ.create();
packet.setAttribute("to", "pubsub.example.com");

Element command = new DefaultElement("command");
command.setXMLNS("http://jabber.org/protocol/commands");
command.setAttribute("node", "create-node");
packet.addChild(command);

Element x = new DefaultElement("x");
x.setXMLNS("jabber:x:data");

command.addChild(x);

def data = new JabberDataElement(x);
data.addTextSingleField("node", "example");
data.addListSingleField("pubsub#node_type", "leaf");

jaxmpp.send(packet, new AsyncCallback() {
    void onError(Stanza responseStanza, tigase.jaxmpp.core.client.XMPPException.
↪ErrorCondition error) throws JaxmppException {
        println "received error during processing request";
    }

    void onSuccess(Stanza responseStanza) throws JaxmppException {
        x = responseStanza.getFirstChild("command").getFirstChid("x");
        data = new JabberDataElement(x);
        def error = data.getField("Error");
        println "command executed with result = " + (error ? "failure, error = " + error.
↪getFieldValue() : "success");
    }

    void onTimeout() {
        println "command timed out"
    }
});

Thread.sleep(30000);
jaxmpp.disconnect();
```

1.5.6 PubSub Node Presence Protocol

Occupant Use Case

Log in to Pubsub Node

To log in to PubSub Node user must send presence to PubSub component with additional information about node:

```
<presence
    from='hag66@shakespeare.lit/pda'
    id='n13mt3l'
    to='pubsub.shakespeare.lit'>
  <pubsub xmlns='tigase:pubsub:1' node='princely_musings'/>
</presence>
```

Component will publish this information in node:

```
<message from='pubsub.shakespeare.lit' to='francisco@denmark.lit' id='foo'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='princely_musings'>
      <item>
        <presence xmlns='tigase:pubsub:1' node='princely_musings' jid='hag66@shakespeare.
↪lit/pda' type='available'/>
      </item>
    </items>
  </event>
</message>
<message from='pubsub.shakespeare.lit' to='bernardo@denmark.lit' id='bar'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='princely_musings'>
      <item>
        <presence xmlns='tigase:pubsub:1' node='princely_musings' jid='hag66@shakespeare.
↪lit/pda' type='available'/>
      </item>
    </items>
  </event>
</message>
```

And then will send notification with presences of all occupants to new occupant.

Log out from PubSub Node

To logout from single node, user must send presence stanza with type unavailable:

```
<presence
    from='hag66@shakespeare.lit/pda'
    type='unavailable'
    to='pubsub.shakespeare.lit'>
  <pubsub xmlns='tigase:pubsub:1' node='princely_musings'/>
</presence>
```

Component will send events to all occupants as described:

```
<message from='pubsub.shakespeare.lit' to='francisco@denmark.lit' id='foo'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='princely_musings'>
      <item>
```

(continues on next page)

(continued from previous page)

```
        <presence xmlns='tigase:pubsub:1' node='princely_musings' jid='hag66@shakespeare.
↪lit/pda' type='unavailable'/>
      </item>
    </items>
  </event>
</message>
```

If component receives presence stanza with type unavailable without specified node, then component will log out user from all nodes he logged before and publish events.

Retrieving list of all Node Subscribers

To retrieve list of node subscribers, node configuration option `tigase#allow_view_subscribers` must be set to true:

```
<iq type='set'
    from='hamlet@denmark.lit/elsinore'
    to='pubsub.shakespeare.lit'
    id='config2'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub#owner'>
    <configure node='princely_musings'>
      <x xmlns='jabber:x:data' type='submit'>
        <field var='FORM_TYPE' type='hidden'>
          <value>http://jabber.org/protocol/pubsub#node_config</value>
        </field>
        <field var='tigase#allow_view_subscribers'><value>1</value></field>
      </x>
    </configure>
  </pubsub>
</iq>
```

When option is enabled, each subscriber may get list of subscribers the same way as owner.

```
<iq type='get'
    from='hamlet@denmark.lit/elsinore'
    to='pubsub.shakespeare.lit'
    id='subman1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub#owner'>
    <subscriptions node='princely_musings'/>
  </pubsub>
</iq>
```

There is extension to filter returned list:

```
<iq type='get'
    from='hamlet@denmark.lit/elsinore'
    to='pubsub.shakespeare.lit'
    id='subman1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub#owner'>
    <subscriptions node='princely_musings'>
        <filter xmlns='tigase:pubsub:1'>
            <jid contains='@denmark.lit' />
```

(continues on next page)

(continued from previous page)

```
        </filter>
    </subscriptions>
  </pubsub>
</iq>
```

In this example will be returned all subscriptions of users from domain “denmark.lit”.

Offline Message Sink

Messages sent to offline users is published in pubsub node, from where that message is sent to all the node subscribers as a pubsub notification.

```
<message from='pubsub.coffeebean.local' to='bard@shakespeare.lit' id='foo'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='message_sink'>
      <item id='ae890ac52d0df67ed7cfdf51b644e901'>
        <message type="chat" xmlns="jabber:client" id="x2ps6u0004"
          to="userB_h6x1bt0002@coffeebean.local"
          from="userA_uyhx8p0001@coffeebean.local/1149352695-tigase-20">
          <body>Hello</body>
        </message>
      </item>
    </items>
  </event>
</message>
```

Configuration

The pubsub node must be created and configured beforehand:

Create node

```
<iq type='set'
    to='pubsub.coffeebean.local'
    id='create1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <create node='message_sink'/>
  </pubsub>
</iq>
```

After that is done, you need to add SessionManager as a publisher:

Add sess-man as publisher

```
<iq type='set'
    to='pubsub.coffeebean.local'
    id='ent2'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub#owner'>
    <affiliations node='message_sink'>
      <affiliation jid='sess-man@coffeebean.local' affiliation='publisher'/>
    </affiliations>
```

(continues on next page)

(continued from previous page)

```
  </pubsub>
</iq>
```

Finally, the 'msgoffline' offline messages processor must be configured as well

config.tdsl configuration

```
sess-man {
    msgoffline () {
        msg-pubsub-jid = 'pubsub.coffeebean.local'
        msg-pubsub-node = 'message_sink'
        msg-pubsub-publisher = 'sess-man@coffeebean.local'
    }
}
```

Usage

Because these sinks use a standard pubsub component, administration of the sink node is identical to any other pubsub node. XEP-0060 defines standard pubsub usage and management.

Managing Subscriptions

Add new Subscriber

```
<iq type='set'
    to='pubsub.coffeebean.local'
    id='subman2'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub#owner'>
    <subscriptions node='message_sink'>
      <subscription jid='bard@shakespeare.lit' subscription='subscribed'/>
    </subscriptions>
  </pubsub>
</iq>
```

Remove Subscriber

```
<iq type='set'
    to='pubsub.coffeebean.local'
    id='subman2'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub#owner'>
    <subscriptions node='message_sink'>
      <subscription jid='bard@shakespeare.lit' subscription='none'/>
    </subscriptions>
  </pubsub>
</iq>
```

1.6 REST API

All example calls to pubsub REST API are prepared for pubsub component running at `pubsub.example.com`. It is required to replace this value with JID of pubsub component from your installation.

It is possible to provide parameters to requests as:

XML
: All parameters passed in content of HTTP request needs to be wrapped with `<data/>` tag as root tag of XML document, while returned parameters will be wrapped `<result/>` tag as root tag of XML document.

JSON
: Parameters must be passed as serialized JSON object. Additionally `Content-Type` header of HTTP request needs to be set to `application/json`.

1.6.1 Create a node

HTTP URI: `/rest/pubsub/pubsub.example.com/create-node`

Available HTTP methods:

GET

Method returns example content which contains all required and optional parameters that may be passed to newly created node.

POST

Command requires fields `node` and `pubsub#node_type` to be filled with proper values for execution.

- `node` - field should contain id of node to create
- `owner` - field may contains jid which should be used as jid of owner of newly created node (will use jid of Tigase HTTP API Component if not passed)
- `pubsub#node_type` - should contain type of node to create (two values are possible: `leaf` - node to which items will be published, `collection` - node which will contain other nodes)

Example content to create node of id `example` and of type `leaf` and with owner set to `admin@example.com`.

Using XML

Request in XML.

```
<data>
  <node>example</node>
  <owner>admin@example.com</owner>
  <pubsub prefix="true">
    <node_type>leaf</node_type>
  </pubsub>
</data>
```

Response in XML.

```
<result>
  <Note type="fixed">
    <value>Operation successful</value>
  </Note>
</result>
```

Using JSON

Request in JSON.

```
{
  "node" : "example",
  "owner" : "admin@example.com",
  "pubsub#node_type" : "leaf"
}
```

Response in JSON.

```
{
  "Note": "Operation successful"
}
```

1.6.2 Delete a node

HTTP URI: `/rest/pubsub/pubsub.example.com/delete-node`

Available HTTP methods:

GET

Method returns example content which contains all required and optional parameters that may be passed.

POST

Command requires field `node` to be filled.

- `node` - field should contain id of node to delete

Example content to delete node with id `example`

Using XML

Request in XML.

```
<data>
  <node>example</node>
</data>
```

Response in XML.

```
<result>
  <Note type="fixed">
    <value>Operation successful</value>
  </Note>
</result>
```

Using JSON

Request in JSON.

```
{
  "node" : "example"
}
```

Response in JSON.

```
{
  "Note" : "Operation successful"
}
```

1.6.3 Subscribe to a node

HTTP URI: `/rest/pubsub/pubsub.example.com/subscribe-node`

Available HTTP methods:

GET

Method returns example content which contains all required and optional parameters that may be passed.

POST

Command requires fields `node` and `jids` to be filled.

- `node` - field should contain id of node to subscribe to
- `jids` - field should contain list of jids to be subscribed to node

Example content to subscribe to node with id `example` users with jid `test1@example.com` and `test2@example.com`

Using XML

Request in XML.

```
<data>
  <node>example</node>
  <jids>
    <value>test1@example.com</value>
    <value>test2@example.com</value>
  </jids>
</data>
```

Response in XML.

```
<result>
  <Note type="fixed">
    <value>Operation successful</value>
  </Note>
</result>
```

Using JSON

Request in JSON.

```
{
  "node" : "example",
  "jids" : [
    "test1@example.com",
    "test2@example.com"
  ]
}
```

Response in JSON.

```
{
  "Note" : "Operation successful"
}
```

1.6.4 Unsubscribe from a node

HTTP URI: `/rest/pubsub/pubsub.example.com/unsubscribe-node`

Available HTTP methods:

GET

Method returns example content which contains all required and optional parameters that may be passed.

POST

Command requires fields `node` and `jids` to be filled.

- `node` - field should contain id of node to unsubscribe from
- `jids` - field should contain list of jids to be unsubscribed from node

Example content to unsubscribe from node with id `example` users `test1@example.com` and `test2@example.com`

Using XML

Request in XML.

```
<data>
  <node>example</node>
  <jids>
    <value>test@example.com</value>
    <value>test2@example.com</value>
  </jids>
</data>
```

Response in XML.

```
<result>
  <Note type="fixed">
    <value>Operation successful</value>
  </Note>
</result>
```

Using JSON

Request in JSON.

```
{
  "node" : "example.com",
  "jids" : [
    "test@example.com",
    "test2@example.com"
  ]
}
```

Response in JSON.

```
{
  "Note" : "Operation successful"
}
```

1.6.5 Publish an item to a node

HTTP URI: `/rest/pubsub/pubsub.example.com/publish-item`

Available HTTP methods:

GET

Method returns example content which contains all required and optional parameters that may be passed.

POST

Command requires fields `node` and `entry` to be filled

- `node` - field should contain id of node to publish to
- `item-id` - field may contain id of entry to publish
- `expire-at` - field may contain timestamp (in XEP-0082 format) after which item should not be delivered to user
- `entry` - field should contain multi-line entry content which should be valid XML value for an item

Example content to publish item with id `item-1` to node with id `example` and with content in example field. P

Using XML

with XML payload

In this example we will use following XML payload:

Payload.

```
<item-entry>
  <title>Example 1</title>
  <content>Example content</content>
</item-entry>
```

Request in XML.

```
<data>
  <node>example</node>
  <item-id>item-1</item-id>
  <expire-at>2015-05-13T16:05:00+02:00</expire-at>
  <entry>
    <item-entry>
      <title>Example 1</title>
      <content>Example content</content>
    </item-entry>
  </entry>
</data>
```

Response in XML.

```
<result>
  <Note type="fixed">
    <value>Operation successful</value>
  </Note>
</result>
```

with JSON payload

It is possible to publish JSON payload as value of published XML element. In example below we are publishing following JSON object:

Payload.

```
{ "key-1" : 2, "key-2" : "value-2" }
```

Request in XML.

```
<data>
  <node>example</node>
  <item-id>item-1</item-id>
  <expire-at>2015-05-13T16:05:00+02:00</expire-at>
  <entry>
    <payload>{ &quot;key-1&quot; : 2, &quot;key-2&quot; : &quot;value-2&quot; }</payload>
  </entry>
</data>
```

Response in XML.

```
<result>
  <Note type="fixed">
    <value>Operation successful</value>
  </Note>
</result>
```

Using JSON

with XML payload

To publish XML using JSON you need to set serialized XML payload as value for `entry` key. In this example we will use following XML payload:

Payload.

```
<item-entry>
  <title>Example 1</title>
  <content>Example content</content>
</item-entry>
```

Request in JSON.

```
{
  "node" : "example",
  "item-id" : "item-1",
  "expire-at" : "2015-05-13T16:05:00+02:00",
  "entry" : "<item-entry>
    <title>Example 1</title>
    <content>Example content</content>
  </item-entry>"
}
```

Response in JSON.

```
{
  "Note" : "Operation successful"
}
```

with JSON payload

As JSON needs to be set as a value of an XML element it will be wrapped on server side as a value for <payload/> element.

Payload.

```
{ "key-1" : 2, "key-2" : "value-2" }
```

Request in JSON.

```
{
  "node" : "example",
  "item-id" : "item-1",
  "expire-at" : "2015-05-13T16:05:00+02:00",
  "entry" : {
    "key-1" : 2,
    "key-2" : "value-2"
  }
}
```

Response in JSON.

```
{
  "Note" : "Operation successful"
}
```

Published item.

```
<payload>{ &amp;quot;key-1&amp;quot; : 2, &amp;quot;key-2&amp;quot; : &amp;quot;value-2&amp;quot; }</payload>
```

1.6.6 Delete an item from a node

HTTP URI: `/rest/pubsub/pubsub.example.com/delete-item`

Available HTTP methods:

GET

Method returns example content which contains all required and optional parameters that may be passed.

POST

Command requires fields `node` and `item-id` to be filled

- `node` - field contains id of node to publish to
- `item-id` - field contains id of entry to publish

Example content to delete an item with id `item-1` from node with id `example`.

Using XML

Request in XML.

```
<data>
  <node>example</node>
  <item-id>item-1</item-id>
</data>
```

Response in XML.

```
<result>
  <Note type="fixed">
    <value>Operation successful</value>
  </Note>
</result>
```

Using JSON

Request in JSON.

```
{
  "node" : "example",
  "item-id" : "item-1"
}
```

Response in JSON.

```
{
  "Note" : "Operation successful"
}
```

1.6.7 List available nodes

HTTP URI: `/rest/pubsub/pubsub.example.com/list-nodes`

Available HTTP methods:

GET

Method returns list of available pubsub nodes for domain passed as part of URI (`pubsub.example.com`).

Example response in XML.

```
<result>
  <title>List of available nodes</title>
  <nodes label="Nodes" type="text-multi">
    <value>test</value>
    <value>node_54idf40037</value>
    <value>node_3ws5lz0037</value>
  </nodes>
</result>
```

in which we see nodes: `test`, `node_54idf40037` and `node_3ws5lz0037`.

Example response in JSON.

```
{
  "title" : "List of available nodes",
  "nodes" : [
    "test",
    "node_54idf40037",
    "node_3ws5lz0037"
  ]
}
```

in which we see nodes: `test`, `node_54idf40037` and `node_3ws5lz0037`.

1.6.8 List published items on node

HTTP URI: `/rest/pubsub/pubsub.example.com/list-items`

Available HTTP methods:

GET

Method returns example content which contains all required and optional parameters that may be passed.

POST

Command requires field `node` to be filled

- `node` - field contains id of node which items we want to list

Example content to list of items published on node with id `example`.

Using XML

Request in XML.

```
<data>
  <node>example</node>
</data>
```

Response in XML.

```
<result>
  <title>List of PubSub node items</title>
  <node label="Node" type="text-single">
    <value>example</value>
  </node>
  <items label="Items" type="text-multi">
    <value>item-1</value>
    <value>item-2</value>
  </items>
</result>
```

where `item-1` and `item-2` are identifiers of published items for node `example`.

Using JSON

Request in JSON.

```
{
  "node" : "example"
}
```

Response in JSON.

```
{
  "title" : "List of PubSub node items",
  "node" : "example",
  "items" : [
    "item-1",
    "item-2"
  ]
}
```

where `item-1` and `item-2` are identifiers of published items for node `example`.

1.6.9 Retrieve item published on node

HTTP URI: `/rest/pubsub/pubsub.example.com/retrieve-item`

Available HTTP methods:

GET

Method returns example content which contains all required and optional parameters that may be passed.

POST

Command requires fields `node` and `item-id` to be filled

- `node` - field contains id of node which items we want to list
- `item-id` - field contains id of item to retrieve

Example content to list of items published on node with id `example`.

Using XML

Request in XML.

```
<data>
  <node>example</node>
  <item-id>item-1</item>
</data>
```

Response in XML.

```
<result>
  <title>Retrieve PubSub node item</title>
  <node label="Node" type="text-single">
    <value>example</value>
  </node>
  <item-id label="Item ID" type="text-single">
    <value>item-1</value>
  </item-id>
  <item label="Item" type="text-multi">
    <value>
      <item expire-at="2015-05-13T14:05:00Z" id="item-1">
        <item-entry>
          <title>Example 1</title>
          <content>Example content</content>
        </item-entry>
      </item>
    </value>
  </item>
</result>
```

inside item element there is XML encoded element which is published on node `example` with id `item-1`.

Using JSON

Request in JSON.

```
{
  "node" : "example",
  "item-id" : "item-1"
}
```

Response in JSON.

```
{
  "title" : "Retrieve PubSub node item",
  "node" : "example",
  "item-id" : "item-1",
  "item" : [
    "<item expire-at\"2015-05-13T14:05:00Z\" id=\"item-1\">
      <item-entry>
        <title>Example 1</title>
        <content>Example content</content>
```

(continues on next page)

(continued from previous page)

```
      </item-entry>
    </item>"
  ]
}
```

1.6.10 Retrieve user subscriptions

HTTP URI: `/rest/pubsub/pubsub.example.com/retrieve-user-subscriptions`

Available HTTP methods:

GET

Method returns example content which contains all required and optional parameters that may be passed.

POST

Command requires field `jid` to be filled.

- `jid` - field contains JID of a user for which we want to retrieve subscriptions
- `node-pattern` - field contains regex pattern to match. When field is not empty, request will return only subscribed nodes which match this pattern. If field should be empty it may be omitted in a request.

Example content to retrieve list of nodes to which user `test@example.com` is subscribed at `pubsub.example.com` which starts with `test-` (pattern `test-.*`)

Using XML

Request in XML.

```
<data>
  <jid>test@example.com</jid>
  <node-pattern>test-.*</node-pattern>
</data>
```

Response in XML.

```
<result>
  <nodes label="Nodes" type="text-multi">
    <value>test-123</value>
    <value>test-342</value>
  </nodes>
</result>
```

Using JSON

Request in JSON.

```
{
  "jid" : "test@example.com",
  "node-pattern" : "test-.*"
}
```

Response in JSON.

```
{
  "nodes" : [
    "test-123",
    "test-342"
  ]
}
```

CHAPTER

TWO

LIMITATIONS

2.1 Addressing

Within Tigase, all pubsub component address MUST be domain-based address and not a JID style address. This was made to simplify communications structure. Tigase will automatically set component names to `pubsub.domain`, however any messages send to `pubsub@domain` will result in a `SERVICE_UNAVAILABLE` error.

Pubsub nodes within Tigase can be found as a combination of JID and node where nodes will be identified akin to service discovery. For example, to address a friendly node, use the following structure:

```
<iq to='pubsub.domain'>
  <query node='friendly node'/>
</iq>
```