

RemoteTM



Copyright © 2008 - 2021 Maxprograms

Table of Contents

Introduction	1
Installation And Configuration	2
Preparation	2
Installation Procedure	2
Email Server Configuration	2
Translation Memories	4
Add Memory	4
Remove Memory	5
Set Access Permissions	5
Import TMX	6
Export TMX	7
Close Memories	8
Refresh Memories	8
Users Management	9
Add User	9
Edit User	11
Remove User	11
Lock/Unlock User	12
Password Management	12
Change Password	12
Password Reset	13
REST API	14
Access Management	14
Authorization Request	15
Logout	15
Memories Management	16
Get Memories	17
Add Memory	17
Remove Memory	18
Import TMX	19
Upload File	19
Import File	20
Export TMX	21
Download File	22
Memory Permissions	22
Get Permissions	22
Set Permissions	23
Open Memory	24
Close Memory	25
Close Memories	26

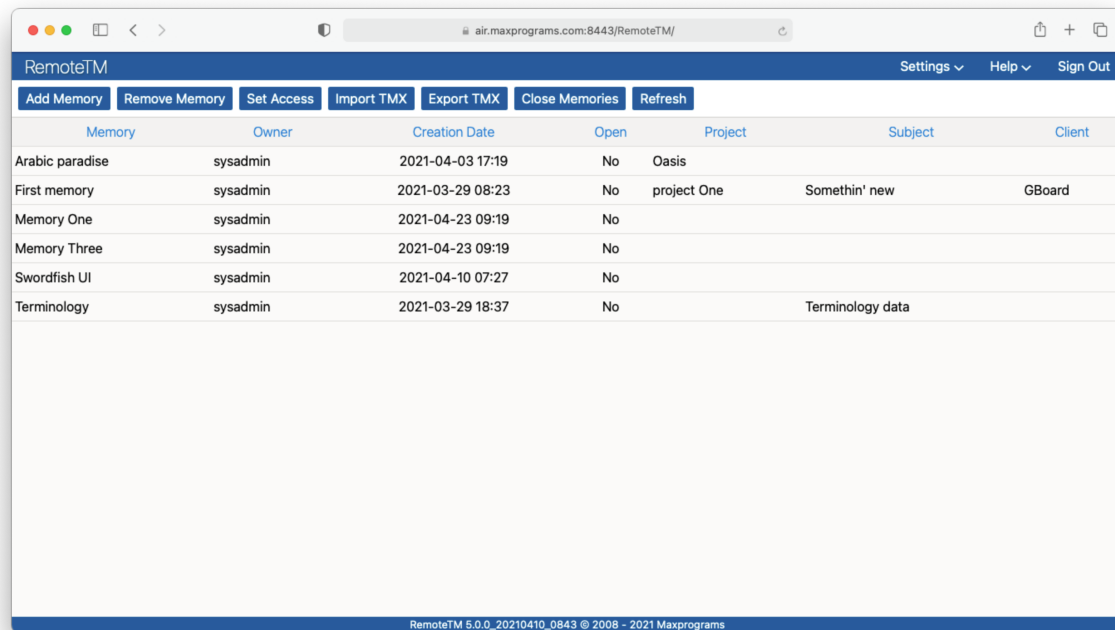
Store Translation Unit	27
Remove Translation Unit	27
Commit	27
Search Translations	27
Search All Translations	27
Concordance Search	27
Batch Translation	27
Get Projects	27
Get Subjects	27
Get Clients	27
Get Memory Projects	27
Get Memory Subjects	27
Get Memory Clients	27
Users Management	27
Get Users	27
Add User	28
Get User	29
Update User	30
Remove User	31
Lock/Unlock User	32
Email Server Management	33
Get Email Server	33
Update Email Server	33

Introduction

RemoteTM is a secure application designed for sharing the [Translation Memory](#) engine used in [Swordfish IV](#) in LAN environments or over the Internet.

RemoteTM runs on top of a Java Servlet container like [Apache Tomcat](#) and can be used in all operating systems where the combination of Java and Apache Tomcat works.

RemoteTM has an open [REST API](#) that allows integration with any translation tool using web services.



The screenshot shows the RemoteTM web application interface. At the top, there is a navigation bar with the title 'RemoteTM' and links for 'Settings', 'Help', and 'Sign Out'. Below the navigation bar is a toolbar with buttons for 'Add Memory', 'Remove Memory', 'Set Access', 'Import TMX', 'Export TMX', 'Close Memories', and 'Refresh'. The main content area displays a table with the following columns: Memory, Owner, Creation Date, Open, Project, Subject, and Client. The table contains six rows of data. At the bottom of the interface, there is a footer with the text 'RemoteTM 5.0.0_20210410_0843 © 2008 - 2021 Maxprograms'.

Memory	Owner	Creation Date	Open	Project	Subject	Client
Arabic paradise	sysadmin	2021-04-03 17:19	No	Oasis		
First memory	sysadmin	2021-03-29 08:23	No	project One	Somethin' new	GBoard
Memory One	sysadmin	2021-04-23 09:19	No			
Memory Three	sysadmin	2021-04-23 09:19	No			
Swordfish UI	sysadmin	2021-04-10 07:27	No			
Terminology	sysadmin	2021-03-29 18:37	No		Terminology data	

Installation And Configuration

The following applications are required to install and run RemoteTM Web Server:

- Java 11, available at <https://www.adoptopenjdk.net>.
- [Apache Tomcat](https://tomcat.apache.org/) available from <https://tomcat.apache.org/>, configured with an SSL certificate to work with HTTPS protocol.

JavaScript must be enabled in all client web browsers for using RemoteTM.

Preparation

Collect the following information before installing RemoteTM Web Server:

- The URL in which the [Apache Tomcat](https://tomcat.apache.org/) server accepts HTTPS requests;
- Configuration details for the SMTP server to use for sending email notifications:
 - Server name
 - Server port
 - User name
 - User password

Installation Procedure

About this task

Follow these steps to install RemoteTM Web Server:

Procedure

1. Stop [Apache Tomcat](https://tomcat.apache.org/) if it is running.
2. Copy `RemoteTM.war` to the `/webapps` folder of Apache Tomcat.
3. Delete the `/webapps/RemoteTM` folder generated by a previous installation if it exists.
4. Start the Apache Tomcat server.
5. Open the server URL in a web browser with `/RemoteTM` appended to it (e.g. if the server URL is `https://myserver.com:8443` then open `https://myserver.com:8443/RemoteTM`)
6. Login with these default credentials: **User Name:** `sysadmin` **Password:** `secData`
7. [Change password](#) for `sysadmin` user.
8. [Configure an email server](#).
9. [Modify the `sysadmin` user](#) and set a real email address.

Email Server Configuration

Follow these steps to configure the SMTP server that RemoteTM uses for sending email.

About this task

RemoteTM sends email notifications in these cases:

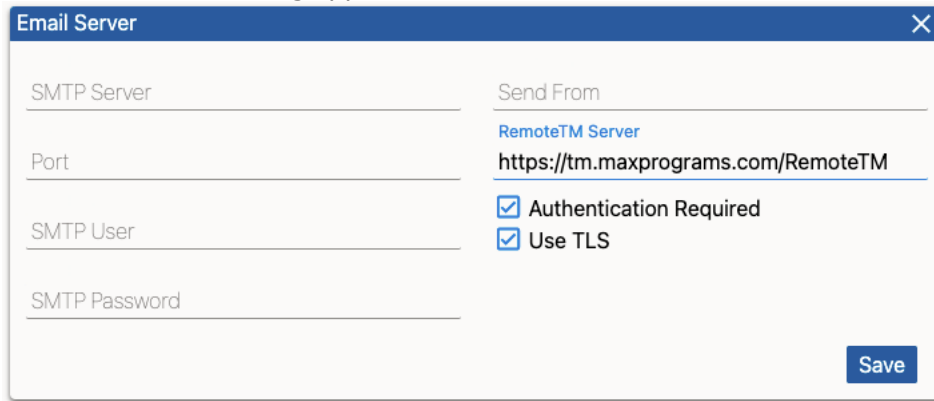
- When a new user is added, to provide new credentials
- When TMX files are imported, to deliver import results

- When a password reset is requested, to provide a custom reset link

Procedure

1. Login as a user with "System Administrator" privileges.
2. Click **Settings** → **Email Server** in main toolbar.

The **Email Server** dialog appears:



The screenshot shows the 'Email Server' dialog box with the following fields and options:

- SMTP Server**: Text input field.
- Port**: Text input field.
- SMTP User**: Text input field.
- SMTP Password**: Text input field.
- Send From**: Text input field containing 'RemoteTM Server' and the URL 'https://tm.maxprograms.com/RemoteTM'.
- Authentication Required**: Check box (checked).
- Use TLS**: Check box (checked).
- Save**: Button.

3. Enter the name or IP of the SMTP server in the **SMTP Server** text input.
4. Enter the port number in which the SMTP server accepts requests in the **Port** text input.
5. Enter the user name for the SMTP server in the **SMTP User** text input.
6. Enter the password for the user selected in previous step in the **SMTP Password** text input.
7. Enter the email address used for sending notifications in the **Send From** text input.
8. If necessary, adjust the URL of the RemoteTM server in the **RemoteTM Server** text input. This field is automatically populated by RemoteTM.
9. Check the **Authentication Required** check box if your SMTP server requires authentication.
10. Check the **Use TLS** check box if your SMTP server requires TLS/SSL protocols.
11. Click the **Save** button.

Results

The updated email server configuration is stored in RemoteTM server.

Translation Memories

Use the buttons on RemoteTM's main toolbar to manage your memories.

[Add Memory](#) [Remove Memory](#) [Set Access](#) [Import TMX](#) [Export TMX](#) [Close Memories](#) [Refresh](#)

The buttons on RemoteTM's main tool bar let you perform these tasks:

- [Add Memory](#)
- [Remove Memory](#)
- [Set Access Permissions](#)
- [Import TMX](#)
- [Export TMX](#)
- [Close Memories](#)
- [Refresh Memories](#)

Note

Make sure that all memories are closed before shutting down or restarting Apache Tomcat.

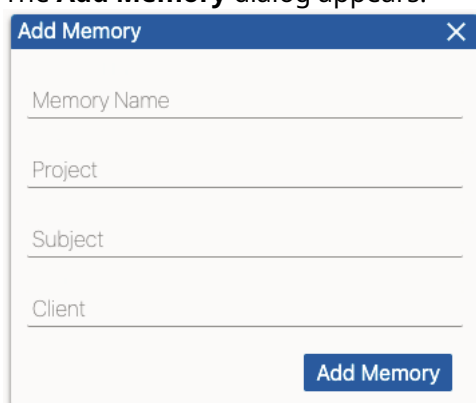
Add Memory

Follow these steps to add a new Translation Memory to RemoteTM.

Procedure

1. Login as a user with "System Administrator" or "Project Manager" privileges.
2. Click the **Add Memory** button on the main toolbar.

The **Add Memory** dialog appears:



3. Type a name for the new memory in the **Memory Name** text box.
Only Latin characters, numbers and underscores should be used in a memory name.
4. Optionally, type a project description in the **Project** text input.
5. Optionally, type a subject for the TM data in the **Subject** text input.

6. Optionally, enter a client name in the **Client** text box.
7. Click the **Add Memory** button.

Results

A translation memory is created with full access granted to the owner.

Remove Memory

Follow these steps to remove a Translation Memory from RemoteTM.

About this task

When you no longer need a Translation Memory, you may remove it from RemoteTM. Backup your data [exporting the memory as TMX](#) before removing it.

Procedure

1. Select a memory on the main list by clicking on it.
2. Click on the **Remove Memory** button on the main toolbar.
3. Confirm memory removal on the dialog displayed by the browser.

CAUTION

Memory removal is permanent and cannot be undone.

Results

Selected memory is removed and the list of memories is updated.

Set Access Permissions

Follow these steps to set the access levels for each user on your memories.

About this task

User access to translation memories content is controlled by setting access permissions. The access rights a user can have for a given translation memory are:

- Read
- Write
- Export

The following table summarizes the actions allowed to a user that has been granted access to a database:

Permission	Allowed Actions
Read	Use the translation memory in Swordfish IV or RemoteTM's REST API for the following tasks: <ul style="list-style-type: none">• TM matches retrieval

Permission	Allowed Actions
	<ul style="list-style-type: none"> • Concordance searches • Terms retrieval • Term searches
Write	<ul style="list-style-type: none"> • Import TMX files using RemoteTM's web interface. • Import TMX files using Swordfish IV or RemoteTM's REST API. • Select the translation memory as write-enabled database in Swordfish IV or RemoteTM's REST API for storing segments at translation time.
Export	<ul style="list-style-type: none"> • Export the translation memory as TMX using RemoteTM's web interface. • Export the database as TMX using Swordfish IV or RemoteTM's REST API.

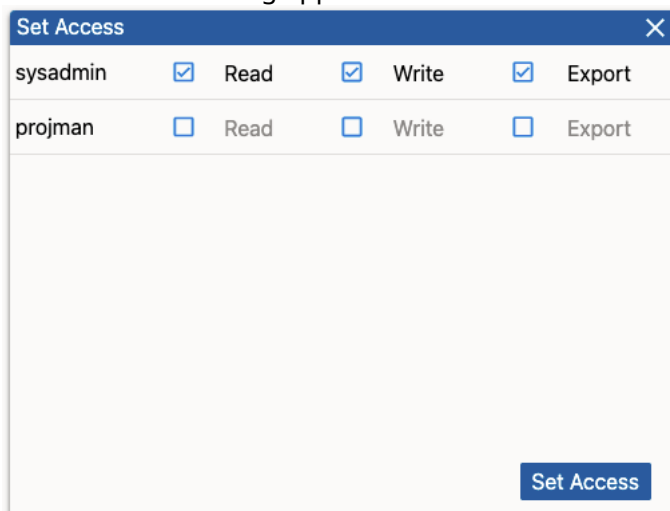
Users with "System Administrator" privileges can set access permissions for any translation memory.

Users with "Project Manager" privileges can set access permissions for any database they own.

Procedure

1. Select a memory on the main list by clicking on it.
2. Click the **Set Access** button on the main toolbar.

The **Set Access** dialog appears:



3. Click on the checkboxes next each user ID to set the desired permission levels for that user.
4. Click on the **Set Access** button.

Import TMX

Steps for importing TMX files into a translation memory using RemoteTM's web interface.

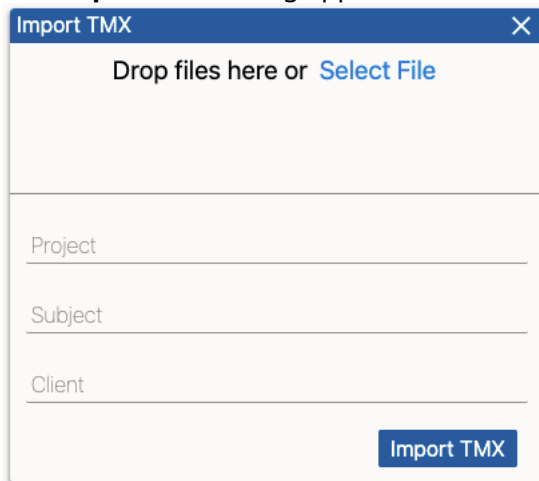
About this task

RemoteTM translation memories can be populated with data from TMX files either using [Swordfish IV](#), its [REST API](#) or RemoteTM's own web interface. Using RemoteTM's web interface is faster.

Procedure

1. Select a memory on the main list by clicking on it.
2. Click the **Import TMX** button on the main toolbar.

The **Import TMX** dialog appears:



3. Drag a TMX file from your computer and drop it on the top area of the **Import TMX** dialog. Alternatively, click on the **Select File** link to open a dialog for selecting a file from your computer.
4. Optionally, type a project description in the **Project** text input.
5. Optionally, type a subject for the TMX data in the **Subject** text input.
6. Optionally, enter a client name in the **Client** text box.
7. Click the **Import TMX** button.

Results

RemoteTM uploads the selected TMX file and starts importing its data into the selected memory. On completion, an email with import results is sent to the user.

Export TMX

Steps for exporting a translation memory as TMX using RemoteTM's web interface.

About this task

RemoteTM memories can be exported as TMX files either using [Swordfish IV](#), its [REST API](#) or RemoteTM's own web interface. Using RemoteTM's web interface is faster.

Procedure

1. Select a memory on the main list by clicking on it.
2. Click the **Export TMX** button.

Results

RemoteTM exports the selected memory as TMX and once the export finishes, the TMX file is downloaded by the browser.

Close Memories

About this task

All translation memories must be closed before shutting down / restarting Apache Tomcat. Closing Apache Tomcat when translation memories are open may result in data loss.

Procedure

1. Click the **Close Memories** button on the main toolbar.
2. Confirm the close operation.

Results

All data is flushed to disk and translation memories are closed. Once all memories are closed, the list of memories is refreshed.

Refresh Memories

About this task

The list of memories and their statuses is loaded when a user signs in and is not updated automatically to reflect operations happening in background. It may be necessary to refresh the list to verify that there aren't open memories before shutting down [Apache Tomcat](#).

Procedure

1. Click the **Refresh Memories** button on the main toolbar.

Results

The list of memories is updated.

Users Management

RemoteTM is a multi-user application. This section provides information for administering RemoteTM users.

A RemoteTM user can have one of these roles:

- System Administrator
- Project Manager
- Translator

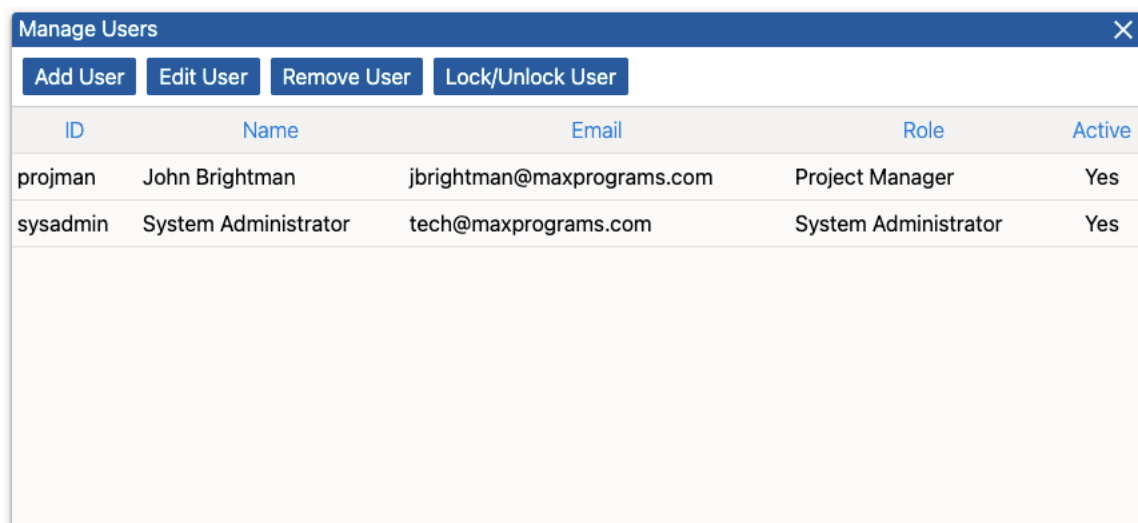
The following table defines task restrictions based on user roles:

Task	System Administrator	Project Manager	Translator
Create/modify users	Yes	No	No
Create translation memories	Yes	Yes	No
Set access permissions	Yes	Yes	No
Delete a translation memory	Yes	No	No
Close memories	Yes	No	No

The following tasks do not have restrictions based on user roles, restrictions are defined at translation memory level by setting [Access Permissions](#) instead:

- Retrieve translations or perform searches using [Swordfish IV](#) or the [REST API](#).
- Import TMX files
- Write segments to the translation memory using [Swordfish IV](#) or the [REST API](#)
- Export translation memory as TMX

Users are created and maintained in the **Manage Users** dialog that is available in **Settings** menu.



Add User

Follow these steps for adding a new RemoteTM user.

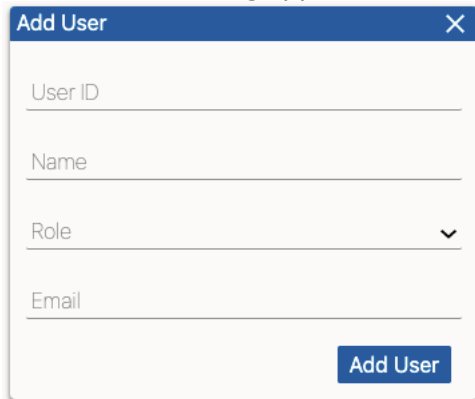
Procedure

1. Click **Settings** → **Manage Users** in main toolbar.

The **Manage Users** dialog appears.

2. In the **Manage Users** dialog, click the **Add User** button.

The **Add User** dialog appears:



3. Type a unique identification code for the new user in the **User ID** text input.
Only Latin characters, numbers, underscores or the '@' sign should be used in a User ID.

4. Type the name of the user in the **Name** text input.

5. Select a role for the user from the **Role** drop-down list. Available options are:

Option	Description
System Administrator	User that can manage the RemoteTM server without restrictions.
Project Manager	User that can create translation memories and assign use rights to other users.
Translator	User with access restricted to assigned translation memories.

6. Type the email address to use for sending notifications to the user in the **Email** text input.

Important

If the email address of the user is invalid, sending the initial login credentials will fail and the user will not be created.

7. Click the **Add User** button.

Results

An email with login credentials is sent to the new user and the user account is created.

Edit User

Follow these steps for modifying an existing RemoteTM user.

About this task

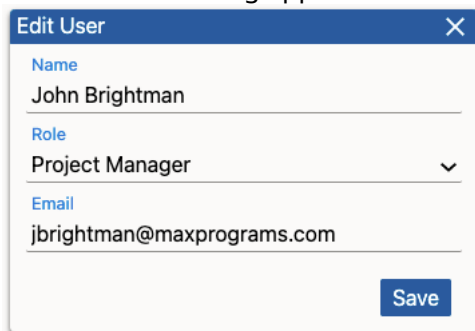
Procedure

1. Click **Settings** → **Manage Users** in main toolbar.

The **Manage Users** dialog appears.

2. Select a user on the **Manage Users** dialog list by clicking on it.
3. In the **Manage Users** dialog, click the **Edit User** button.

The **Edit User** dialog appears:



4. Adjust the contents of **Name**, **Role** or **Email** fields as needed.
5. Click the **Save** button.

Results

The user account is updated with the new data.

Remove User

About this task

You should only delete users that do not own translation memories. If a user owns a translation memory, [lock the user's account](#) instead.

Procedure

1. Click **Settings** → **Manage Users** in main toolbar.

The **Manage Users** dialog appears.

2. Select the user to be removed in the table displayed by the **Manage Users** dialog by clicking on it.
3. Click the **Remove User** button.
4. Confirm the removal operation.

CAUTION

User removal is permanent and cannot be undone.

Results

The selected user account is removed.

Lock/Unlock User

A user account can be temporarily disabled without removing the associated data using this option.

About this task**Procedure**

1. Click **Settings** → **Manage Users** in main toolbar.

The **Manage Users** dialog appears.

2. Select the user to be deleted in the table displayed by the **Manage Users** dialog by clicking on it.
3. Click the **Lock/Unlock User** button.

Results

User status is changed and its status is updated in the Active column.

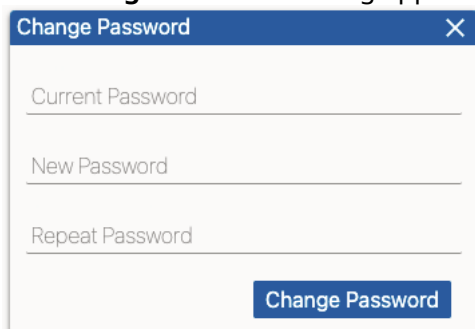
Password Management

Change Password

About this task**Procedure**

1. Click **Settings** → **Change Password** in main toolbar.

The **Change Password** dialog appears:

A screenshot of a 'Change Password' dialog box. It has a blue title bar with the text 'Change Password' and a close button (X). The dialog contains three text input fields: 'Current Password', 'New Password', and 'Repeat Password'. At the bottom right, there is a blue button labeled 'Change Password'.

2. Type your password in the **Current Password** text input.

3. Type a new password in the **New Password** text input.
4. Reenter the new password in the **Repeat Password** text input.
5. Click the **Change Password** button.

Results

On success, the login screen is presented and you need to login again, using the new password, to continue working on RemoteTM.

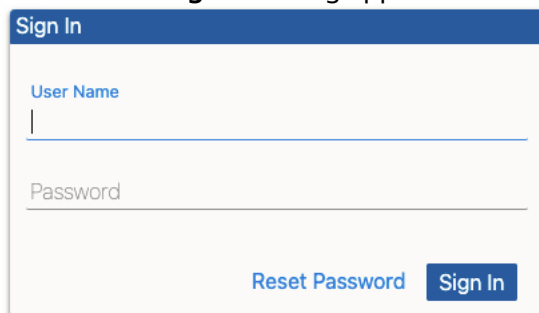
Password Reset

About this task

Procedure

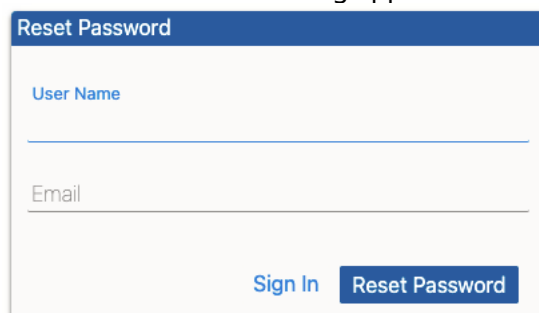
1. Open RemoteTM's web page using your browser.

RemoteTM's **Sign In** dialog appears:

A screenshot of the 'Sign In' dialog box. It has a blue header bar with the text 'Sign In'. Below the header, there are two text input fields: 'User Name' and 'Password'. At the bottom right, there are two buttons: 'Reset Password' (a blue link) and 'Sign In' (a blue button).

2. Click the **Reset Password** link.

The **Reset Password** dialog appears:

A screenshot of the 'Reset Password' dialog box. It has a blue header bar with the text 'Reset Password'. Below the header, there are two text input fields: 'User Name' and 'Email'. At the bottom right, there are two buttons: 'Sign In' (a blue link) and 'Reset Password' (a blue button).

3. Type your user name in the **User Name** text input.
4. Type your email address in the **Email** text input.
5. Click the **Reset Password** button.

Results

If your user name and email address match RemoteTM's records, a message with a custom link for resetting your password will be sent to the indicated email address.

REST API

The **REST** methods exposed by RemoteTM are:

- Access Management
 - Authorization Request
 - Logout
- Memories
 - Get Memories
 - Add Memory
 - Remove Memory
 - Import TMX
 - Upload File
 - Import File
 - Export TMX
 - Download File
- Memory Permissions
 - Get Permissions
 - Set Permissions
- Users Management
 - Get Users
 - Add User
 - Get User
 - Update User
 - Remove User
 - Lock/Unlock User
- Email Server
 - Get Email Server
 - Update Email Server

Access Management

The **REST** methods for access management exposed by RemoteTM are:

- Authorization Request
- Logout

Authorization Request

End point: [RemoteTM URL]/remote

Send a GET request with these headers:

Header	Value
Content-Type	application/json
Authorization	BASIC authentication code

The value part of Authorization header is constructed as follows:

1. Combine `username` + ':' (colon character) + `password` into a single string. Notice that the username field cannot contain a colon.
2. Encode the string generated in step 1 using Base64.
3. Prepend "BASIC " to the string encoded in step 2.

For example, if the value of `username` is `services` and the `password` is `magic#123`, the string to be encoded using Base64 is `services:magic#123`. Then, the header to be sent to RemoteTM is:

```
Authorization: BASIC c2VydmJjZXM6bWFnaWMjMTIz
```

RemoteTM responds with a JSON object.

On success, the value of "status" field is "OK", "ticket" field contains the authorization code issued by RemoteTM and "role" field contains the user's role. Example:

```
{
  "status": "OK",
  "ticket": "28ceb966-150e-4b27-b603-c244160da5b7",
  "role": "PM"
}
```

On error, field "status" is set to "Error" and failure reason is indicated in the "reason" field. Example:

```
{
  "status": "Error",
  "reason": "Access Denied"
}
```

Logout

End point: [RemoteTM URL]/logout

Send a GET request with these headers:

Header	Value
Session	The ticket received from Authorization Request
Content-Type	application/json

RemoteTM responds with a JSON object.

On success, field 'status' is set to 'OK'. Example:

```
{
  "status": "OK"
}
```

On error, field 'status' is set to 'Error'. Example:

```
{
  "status": "Error"
}
```

Memories Management

The REST methods for managing memories exposed by RemoteTM are:

- [Get Memories](#)
- [Add Memory](#)
- [Remove Memory](#)
- [Import TMX](#)
 - [Upload File](#)
 - [Import File](#)
- [Export TMX](#)
 - [Download File](#)
- [Memory Permissions](#)
 - [Get Permissions](#)
 - [Set Permissions](#)
- [Open Memory](#)
- [Close Memory](#)
- [Close Memories](#)
- [Store Translation Unit](#)
- [Remove Translation Unit](#)
- [Commit](#)
- [Search Translations](#)
- [Search All Translations](#)
- [Concordance Search](#)
- [Batch Translation](#)
- [Get Projects](#)
- [Get Subjects](#)
- [Get Clients](#)

- [Get Memory Projects](#)
- [Get Memory Subjects](#)
- [Get Memory Clients](#)

Get Memories

End point: [RemoteTM URL]/memories

Send a GET request with these headers:

Header	Value
Session	The ticket received from Authorization Request
Content-Type	application/json

RemoteTM responds with a JSON object.

On success, field 'status' is set to 'OK' and a list of memories in JSON format is included in field 'memories'. Example:

```
{
  "memories": [{
    "owner": "sysadmin",
    "subject": "Somethin' new",
    "name": "First memory",
    "project": "project One",
    "client": "Greenland",
    "id": "1617017020862",
    "creationDate": "2021-03-29 08:23",
    "open": false
  }, {
    "owner": "sysadmin",
    "subject": "Terminology data",
    "name": "Terminology",
    "project": "",
    "client": "",
    "id": "1617053861912",
    "creationDate": "2021-03-29 18:37",
    "open": false
  }],
  "status": "OK"
}
```

On error, field 'status' is set to 'Error' and field 'reason' contains the error cause. Example:

```
{
  "status": "Error",
  "reason": "Access denied"
}
```

Add Memory

End point: [RemoteTM URL]/memories

Send a `POST` request with these headers:

Header	Value
Session	The ticket received from Authorization Request
Content-Type	application/json

Include these parameters in a JSON body:

Field	Value
command	addMemory
owner	ID of the user that owns the memory
name	The name of the new memory
project	Optional description of a related project
subject	Optional description of the related subject
client	Optional description of a related client

Example:

```
{
  "command": "addMemory",
  "name": "Terminology",
  "owner": "sysadmin",
  "project": "",
  "subject": "Terminology data",
  "client": ""
}
```

RemoteTM responds with a JSON object.

On success, field 'status' is set to 'OK'. Example:

```
{
  "status": "OK"
}
```

On error, field 'status' is set to 'Error' and field 'reason' contains the error cause. Example:

```
{
  "status": "Error",
  "reason": "Access denied"
}
```

Remove Memory

End point: [RemoteTM URL]/memories

Send a `POST` request with these headers:

Header	Value
Session	The ticket received from Authorization Request
Content-Type	application/json

Include these parameters in a JSON body:

Field	Value
command	removeMemory
memory	ID of the memory to remove

Example:

```
{
  "command": "removeMemory",
  "memory": "1617053861912"
}
```

RemoteTM responds with a JSON object.

On success, field 'status' is set to 'OK'. Example:

```
{
  "status": "OK"
}
```

On error, field 'status' is set to 'Error' and field 'reason' contains the error cause. Example:

```
{
  "status": "Error",
  "reason": "Access denied"
}
```

Import TMX

Importing a TMX file into a memory is a two-step process:

1. [Upload File](#)
2. [Import File](#)

Upload File

End point: [RemoteTM URL]/memories

Send a POST request with these headers:

Header	Value
Session	The ticket received from Authorization Request
Content-Type	<ul style="list-style-type: none"> • application/zip to upload the TMX file zipped in request body • multipart/form-data to upload the TMX file as HTML form attachment

Include the TMX file in the request body, either zipped or as HTML form attachment.

RemoteTM tries to store the uploaded file in a temporary location and returns a JSON object.

On success, field 'status' is set to 'OK' and field 'file' contains the temporary file location. Example:

```
{
  "status": "OK",
  "file": "uploaded.tmx"
}
```

On error, field 'status' is set to 'Error' and field 'reason' contains the error cause. Example:

```
{
  "reason": "File upload error",
  "status": "Error"
}
```

Import File

End point: [RemoteTM URL]/memories

Send a POST request with these headers:

Header	Value
Session	The ticket received from Authorization Request
Content-Type	application/json

Include these parameters in a JSON body:

Parameter	Value
command	importTMX
memory	ID of the memory to populate
file	Value of the "file" field received when uploading a TMX file
project	Optional description of a related project
subject	Optional description of the related subject
client	Optional description of a related client
close	Boolean value indicating whether the memory should be closed after importing the TMX file.

Example:

```
{
  "command": "importTMX",
  "memory": "1617053861912",
  "file": "uploaded.tmx",
  "project": "Book Translation",
  "subject": "Chapter One",
}
```

```
{
  "client": "",
  "close": true
}
```

RemoteTM imports the TMX file into the selected memory and sends an email with import results to the session owner.

Export TMX

End point: [RemoteTM URL]/memories

Send a POST request with these headers:

Header	Value
Session	The ticket received from Authorization Request
Content-Type	application/json

Include these parameters in a JSON body:

Field	Value
command	exportMemory
memory	ID of the memory to export
srcLang	The code of the language to set as source or '*all*' if any language is to be treated as source language
close	Boolean value indicating whether the memory should be closed after exporting.

Example:

```
{
  "command": "exportMemory",
  "memory": "1617053861912",
  "srcLang": "*all*",
  "close": false
}
```

RemoteTM responds with a JSON object.

On success, field 'status' is set to 'OK' and the location of the exported file is returned on the 'file' field. Example:

```
{
  "status": "OK",
  "file": "exportedFile.tmx"
}
```

On error, field 'status' is set to 'Error' and field 'reason' contains the error cause. Example:

```
{
  "status": "Error",
```



```
"reason": "Access denied"
}
```

After the memory has been exported to the server filesystem, send a [file download](#) request to RemoteTM.

Download File

End point: [RemoteTM URL]/download

Send a `GET` request with these parameters:

Parameter	Value
Session	The ticket received from Authorization Request
file	URL-encoded name of the file to download

Example:

```
https://myserver.com/RemoteTM/download?session=9321f-1070ae8&file=exportedFile.tmx
```

On success, RemoteTM responds with the requested file as attachment.

Memory Permissions

The REST methods for managing memory permissions exposed by RemoteTM are:

- [Get Permissions](#)
- [Set Permissions](#)

Get Permissions

End point: [RemoteTM URL]/permissions

Send a `GET` request with these headers:

Header	Value
Session	The ticket received from Authorization Request
Content-Type	application/json

Include this parameter in the request URL:

Parameter	Value
id	ID of the memory to query

Example:

```
https://tm.mydomain.com:8443/RemoteTM/permissions?id=1619955225759
```

RemoteTM responds with a JSON object.

On success, field 'status' is set to 'OK' and memory access rights are listed in the 'permissions' field. Example:

```
{
  "permissions": [{
    "memory": "1619955225759",
    "read": true,
    "user": "projman",
    "write": true,
    "export": true
  }, {
    "memory": "1619955225759",
    "read": true,
    "user": "linguist",
    "write": false,
    "export": false
  }, {
    "memory": "1619955225759",
    "read": true,
    "user": "sysadmin",
    "write": true,
    "export": true
  }],
  "status": "OK"
}
```

On error, field 'status' is set to 'Error' and field 'reason' contains the error cause. Example:

```
{
  "reason": "Invalid memory",
  "status": "Error"
}
```

Set Permissions

End point: [RemoteTM URL]/permissions

Send a POST request with these headers:

Header	Value
Session	The ticket received from Authorization Request
Content-Type	application/json

Include these parameters in a JSON body:

Field	Value
memory	ID of the memory to update
permissions	Array containing individual access rights, in the format produced by Get Permissions method

Example:

```
{
  "memory": "1619955225759",
```

```

    "permissions": [{
      "memory": "1619955225759",
      "read": true,
      "user": "projman",
      "write": true,
      "export": true
    }, {
      "memory": "1619955225759",
      "read": true,
      "user": "sysadmin",
      "write": true,
      "export": true
    }, {
      "memory": "1619955225759",
      "read": true,
      "user": "linguist",
      "write": true,
      "export": false
    }
  ]
}

```

RemoteTM responds with a JSON object.

On success, field 'status' is set to 'OK'. Example:

```

{
  "status": "OK"
}

```

On error, field 'status' is set to 'Error' and field 'reason' contains the error cause. Example:

```

{
  "status": "Error",
  "reason": "Access denied"
}

```

Open Memory

End point: [RemoteTM URL]/memories

Send a POST request with these headers:

Header	Value
Session	The ticket received from Authorization Request
Content-Type	application/json

Include these parameters in a JSON body:

Parameter	Value
command	openMemory
memory	ID of the memory to open

Example:

```
{
  "command": "openMemory",
  "memory": "1617053861912"
}
```

RemoteTM responds with a JSON object.

On success, field 'status' is set to 'OK'. Example:

```
{
  "status": "OK"
}
```

On error, field 'status' is set to 'Error' and field 'reason' contains the error cause. Example:

```
{
  "status": "Error",
  "reason": "Access denied"
}
```

Close Memory

End point: [RemoteTM URL]/memories

Send a POST request with these headers:

Header	Value
Session	The ticket received from Authorization Request
Content-Type	application/json

Include these parameters in a JSON body:

Parameter	Value
command	closeMemory
memory	ID of the memory to close

Example:

```
{
  "command": "closeMemory",
  "memory": "1617963861912"
}
```

RemoteTM responds with a JSON object.

On success, field 'status' is set to 'OK'. Example:

```
{
  "status": "OK"
}
```

On error, field 'status' is set to 'Error' and field 'reason' contains the error cause. Example:

```
{
  "status": "Error",
  "reason": "Access denied"
}
```

Close Memories

End point: [RemoteTM URL]/memories

Send a POST request with these headers:

Header	Value
Session	The ticket received from Authorization Request
Content-Type	application/json

Include this parameter in a JSON body:

Parameter	Value
command	closeMemories

Example:

```
{
  "command": "closeMemories"
}
```

RemoteTM closes all open memories.

Store Translation Unit

Remove Translation Unit

Commit

Search Translations

Search All Translations

Concordance Search

Batch Translation

Get Projects

Get Subjects

Get Clients

Get Memory Projects

Get Memory Subjects

Get Memory Clients

Users Management

The REST methods for managing users exposed by RemoteTM are:

- [Get Users](#)
- [Add User](#)
- [Get User](#)
- [Update User](#)
- [Remove User](#)
- [Lock/Unlock User](#)

Get Users

End point: [RemoteTM URL]/users

Send a `GET` request with these headers:

Header	Value
Session	The ticket received from Authorization Request

Header	Value
Content-Type	application/json

RemoteTM responds with a JSON object.

On success, field 'status' is set to 'OK' and a list of users in JSON format is returned in field 'users'.

Example:

```
{
  "users": [{
    "role": "PM",
    "name": "Project Manager",
    "active": true,
    "id": "projman",
    "updated": false,
    "email": "projman@mydomain.com"
  }, {
    "role": "TR",
    "name": "Linguist",
    "active": true,
    "id": "linguist",
    "updated": true,
    "email": "linguist@mydomain.com"
  }, {
    "role": "SA",
    "name": "System Administrator",
    "active": true,
    "id": "sysadmin",
    "updated": true,
    "email": "sysadmin@mydomain.com"
  }],
  "status": "OK"
}
```

On error, field 'status' is set to 'Error' and field 'reason' contains the error cause. Example:

```
{
  "status": "Error",
  "reason": "Access denied"
}
```

Add User

End point: [RemoteTM URL]/users

Send a POST request with these headers:

Header	Value
Session	The ticket received from Authorization Request
Content-Type	application/json

Include these parameters in a JSON body:

Field	Value
command	addUser
id	The ID of the new user
name	The name of the new user
role	'SA', 'PM' or 'TR'
email	The email address of the new user

Example:

```
{
  "command": "addUser",
  "id": "manager",
  "name": "Manager User",
  "role": "PM",
  "email": "pm@mydomain.com"
}
```

RemoteTM responds with a JSON object.

On success, field 'status' is set to 'OK'. Example:

```
{
  "status": "OK"
}
```

On error, field 'status' is set to 'Error' and field 'reason' contains the error cause. Example:

```
{
  "status": "Error",
  "reason": "Duplicated ID"
}
```

An email with a temporary password is sent to the new user's email. The user must login to RemoteTM and set a new password.

Note

User creation fails if RemoteTM's email server is not configured.

Get User

End point: [RemoteTM URL]/users

Send a POST request with these headers:

Header	Value
Session	The ticket received from Authorization Request
Content-Type	application/json

Include these parameters in a JSON body:

Field	Value
command	getUser
id	The ID of the user to get

Example:

```
{
  "command": "getUser",
  "id": "manager"
}
```

RemoteTM responds with a JSON object.

On success, field 'status' is set to 'OK' and requested user data is included in field 'user'. Example:

```
{
  "user": {
    "role": "PM",
    "name": "Principal Manager",
    "active": true,
    "id": "manager",
    "updated": true,
    "email": "pm@mydomain.com"
  },
  "status": "OK"
}
```

On error, field 'status' is set to 'Error' and field 'reason' contains the error cause. Example:

```
{
  "status": "Error",
  "reason": "Access denied"
}
```

Update User

End point: [RemoteTM URL]/users

Send a POST request with these headers:

Header	Value
Session	The ticket received from Authorization Request
Content-Type	application/json

Include these parameters in a JSON body:

Field	Value
command	updateUser
id	The ID of the user to update

Field	Value
name	Updated user name
role	'SA', 'PM' or 'TR'
email	Updated email

Example:

```
{
  "command": "updateUser",
  "id": "manager",
  "name": "Principal Manager",
  "role": "PM",
  "email": "pm@mydomain.com"
}
```

RemoteTM responds with a JSON object.

On success, field 'status' is set to 'OK'. Example:

```
{
  "status": "OK"
}
```

On error, field 'status' is set to 'Error' and field 'reason' contains the error cause. Example:

```
{
  "status": "Error",
  "reason": "Unknown user"
}
```

Remove User

End point: [RemoteTM URL]/users

Send a POST request with these headers:

Header	Value
Session	The ticket received from Authorization Request
Content-Type	application/json

Include these parameters in a JSON body:

Field	Value
command	removeUser
id	The ID of the user to remove

Example:

```
{
  "command": "removeUser",
  "id": "manager"
}
```

```
{
  "id": "manager"
}
```

RemoteTM responds with a JSON object.

On success, field 'status' is set to 'OK'. Example:

```
{
  "status": "OK"
}
```

On error, field 'status' is set to 'Error' and field 'reason' contains the error cause. Example:

```
{
  "status": "Error",
  "reason": "User owns memories"
}
```

Lock/Unlock User

End point: [RemoteTM URL]/users

Send a POST request with these headers:

Header	Value
Session	The ticket received from Authorization Request
Content-Type	application/json

Include these parameters in a JSON body:

Field	Value
command	toggleLock
id	The ID of the user to lock/unlock

Example:

```
{
  "command": "toggleLock",
  "id": "manager"
}
```

RemoteTM responds with a JSON object.

On success, field 'status' is set to 'OK'. Example:

```
{
  "status": "OK"
}
```

On error, field 'status' is set to 'Error' and field 'reason' contains the error cause. Example:

```
{
  "status": "Error",
```

```
{
  "reason": "Unknown user"
}
```

Email Server Management

The REST methods for managing its email server exposed by RemoteTM are:

- [Get Email Server](#)
- [Update Email Server](#)

Get Email Server

End point: [RemoteTM URL]/email

Send a GET request with these headers:

Header	Value
Session	The ticket received from Authorization Request
Content-Type	application/json

RemoteTM responds with a JSON object.

On success, field 'status' is set to 'OK' and server configuration is returned. Example:

```
{
  "server": "smtp.mydomain.com",
  "password": "pass123$",
  "instance": "https://tm.mydomain.com:8443/RemoteTM",
  "authenticate": true,
  "port": "465",
  "from": "remotetm@mydomain.com",
  "tls": false,
  "user": "postmaster",
  "status": "OK"
}
```

On error, field 'status' is set to 'Error' and field 'reason' contains the error cause. Example:

```
{
  "status": "Error",
  "reason": "Access denied"
}
```

Update Email Server

End point: [RemoteTM URL]/email

Send a POST request with these headers:

Header	Value
Session	The ticket received from Authorization Request

Header	Value
Content-Type	application/json

Include these parameters in a JSON body:

Field	Value
server	Name or IP of the SMTP server
port	Port number in which the SMTP server accepts requests
user	User name for the SMTP server
password	Password for the SMTP user
instance	URL of the RemoteTM server
from	Email address used for sending notifications
authenticate	Boolean value indicating whether SMTP server requires authentication
tls	Boolean value indicating whether SMTP server requires TLS/SSL protocols

Example:

```
{
  "server": "smtp.mydomain.com",
  "port": "465",
  "user": "postmaster",
  "password": "pass123$",
  "instance": "https://tm.mydomain.com:8443/RemoteTM",
  "from": "remotetm@mydomain.com",
  "authenticate": true,
  "tls": false
}
```

RemoteTM responds with a JSON object.

On success, field 'status' is set to 'OK'. Example:

```
{
  "status": "OK"
}
```

On error, field 'status' is set to 'Error' and field 'reason' contains the error cause. Example:

```
{
  "status": "Error",
  "reason": "Access denied"
}
```

Glossary

Apache Tomcat

[Apache Tomcat](#) is an open source software implementation of the Java Servlet and Java Server Pages technologies. The Java Servlet and JavaServer Pages specifications are developed under the Java Community Process.

REST

Representational state transfer (REST) is a software architectural style which uses a subset of HTTP. It is commonly used to create interactive applications that use Web services.

Swordfish Translation Editor

Swordfish is a CAT (Computer Aided Translation) tool based on Open Standards that supports MS Office, DITA, HTML and other document formats. Swordfish is published by [Maxprograms](#).

TMX

Translation Memory eXchange (TMX) is an open standard originally published by LISA (Localization Industry Standards Association). The purpose of TMX is to allow easier exchange of translation memory data between tools and/or translation vendors with little or no loss of critical data during the process.

Translation Memory

Translation Memory (TM) is a language technology that enables the translation of segments (paragraphs, sentences or phrases) of documents by searching for similar segments in a database and suggesting matches that are found in the databases as possible translations.