**TraceConnect**

V1.0

Documentation version 1.0.0

# Sommaire

[Contents 2](#_Toc368150643)

[1. Introduction 3](#_Toc368150644)

[2. Prise en main 4](#_Toc368150645)

[2.1. Déploiement TConnect 4](#_Toc368150646)

[2.2. Installation et exécution KTBS 4](#_Toc368150647)

[2.3. Installation le service d’assistance 4](#_Toc368150648)

[2.4. Enregistrer et collecter les éléments observés 4](#_Toc368150649)

[2.5. Authentification et Synchronisation session utilisateur 5](#_Toc368150650)

[2.5.1. Partager la clé secrète 6](#_Toc368150651)

[2.5.2. Créer une nouvelle session utilisateur 6](#_Toc368150652)

[2.5.3. Synchronisation utilisateur 7](#_Toc368150653)

[3. Références API 8](#_Toc368150654)

[3.1. Trace Service (tService) 8](#_Toc368150655)

[3.1.1. trace\_open (options) 8](#_Toc368150656)

[3.1.2. trace\_put\_obsels (options) 8](#_Toc368150657)

[3.1.3. trace\_get\_obsels(options) 8](#_Toc368150658)

[3.2. Assistance Service (tAssistance) 9](#_Toc368150659)

[3.2.1. obsel\_serialize\_html (options) 9](#_Toc368150660)

[3.2.2. draw\_obsels (options) 9](#_Toc368150661)

[3.3. Assistance Server 9](#_Toc368150662)

[3.3.1. User 9](#_Toc368150663)

[3.3.2. Style 10](#_Toc368150664)

[3.4. Trace extension for application (client) : tApp/service.js 10](#_Toc368150665)

[3.4.1. Ouvrir une nouvelle fenêtre assistance 10](#_Toc368150666)

[3.5. Trace extension for application (server) : tApp/service.php 11](#_Toc368150667)

[3.5.1. Synchronisation session utilisateur 11](#_Toc368150668)

# Introduction

**TraceConnect** ou **TConnect** est un ensemble de packages destiné aux développeurs souhaitant intégrer un *service de traces modélisées à* leurs applications. Ces packages permettent de constituer une base de traces d’interactions issues de l’usage de leur application et de faire appel à un assistant chargé de visualiser, exploiter, transformer, naviguer dans la base de traces modélisées ainsi constituées. L’intégration de fonctions d’assistance dans l’application fait l’objet d’une API particulière qui sera utilisé par l’assistant générique, mais qui peut donc être utilisée dans toute application désirant intégrer ses propres modalités d’assistance.

TraceConnect est organisé en trois API:

* tService
  + API pour la collecte et le stockage des interactions sur un dépôt de type SGBT (Système de Gestion de Base de Traces). Ce service permet également l'accès aux traces modélisées en JavaScript.
* tApp
  + API pour les requêtes non-trace nécessaires pour maintenir le lien entre l’application end-user et les autres systèmes.
* tAssistance
  + API pour les requêtes de haut niveau sur les traces (transformation, la visualisation, l'analyse, l'exploitation). Cette API est



Figure Schéma illustrant l'usage des API sur l'environnement OZALID : l'application cliente utilise l'API tService pour enregistrer sur un SGBT (le kTBS du Liris dans le cas précis) les observés collectés; Elle délègue à un assistant générique le soin de visualiser, transformer, naviguer… l’expérience ainsi tracée. L’assistant générique illustre les possibilités de l’API tAssistance, tout en utilisant l’API tService pour les requêtes au SGBT (ici le kTBS).

# Prise en main

## Déploiement TConnect

Le paradigme « assistance à base de trace » intègre trois facettes Application collectant les interactions – SGBT (système de gestion de base de traces) assurant leur gestion, les opérations de base sur les traces (transformations et requêtes) et leur stockage – Assistance : intégrant des fonctionnalités d’usage des traces en facilitant l’interprétation, la réutilisation et le partage. Ces facettes peuvent être réalisées facilement avec les composants de TConnect.

Dans la suite, le SGBT considéré est le « kTBS » tel qu’il est développé et mis à disposition par le LIRIS. Les consignes d’installation du SGBT sont donc celles de ce kTBS. Il convient de substituer cette partie par des consignes adaptées si le SGBT utilisé n’était pas le kTBS. Les principes de fonctionnement des API proposées ne changeraient pas (s’il s’agit bien de SGBT respectant la théorie de la trace modélisée et de SGBT).

## Installation et exécution KTBS

Premièrement, il faut disposer d’un système de gestion de base de traces en ligne. Le système de gestion de base de traces par défaut est le KTBS. Pour installer et exécuter le KTBS, suivre le guide <https://kernel-for-trace-based-systems.readthedocs.org/en/latest/tutorials/install.html>

Après la configuration et l’exécution, vous aurez l’URL de votre kTBS ressemblant à [http://*your.server.name*/ktbs/](http://your.server.name/ktbs/). (your.server.name est la machine surlaquelle vous avez pu installer le kTBS)

## Installation du service d’assistance

Deuxièmement, vous voudriez avoir un service d’assistance en ligne (cas où vous ne gérez pas l’assistance dans votre application). Le package Assistance (tAssistance) est écrit en PHP et JavaScript. Donc, vous devez avoir un serveur en ligne avec le mode PHP5 activé. Ensuite, vous mettez tout le package tAssistance dans le dossier web root (par exemple : sous Ubuntu « /var/www/ ») et activez le dossier tAssistance comme un nouveau site. Le serveur abritant le service d’assistance peut être naturellement différent (ou non) du serveur abritant les service kTBS.

Après la configuration et l’exécution, vous aurez l’URL de votre service d’assistance ressemblant à <http://your.assistance-server.name/tAssistance/>.

## Enregistrer et collecter les éléments observés dans votre application.

Après installation du KTBS et déploiement du service d’assistance en ligne, vous pourrez enregistrer les éléments observés dans le kTBS et faire appel à l’assistance pour visualiser, transformer, … les traces modélisées ainsi disponibles.

Pour faire cela, vous devez déclarer le package tService dans l’entête de toutes les pages de l’application dont vous voulez collecter les traces d’interaction sous forme de M-Traces.

**01** <html>

**02** <head>

**03** <meta http-equiv=**"Content-Type"** content=**"text/html; charset=UTF-8"**>

**04** <title>**Application 1**</title>

**05** <link href=**"css/trace.css"** rel=**"stylesheet"** >

**06** <script type=**"text/javascript"** src=**"your\_path\_tService/js/js.php"**></script>

**07**

**08** </head>

**09** <body>

**10 …**

**11** <div>

Exemple : En-tête des pages applicatives collectant des M-Traces

Ensuite, vous pourrez utiliser le package tService afin de créer une nouvelle trace et collecter les éléments observés (obsels). Par exemple :

**01** <div>

**02 Demo buttons**

**03** <button id=**"bta"** onclick=**"click\_a()"**>**A**</button>

**04** <button id=**"bta"**>**B**</button>

**05** <button id=**"bta"**>**C**</button>

**06** </div>

**07** <script type=**"text/javascript"**>

08 ***function*** click\_a**(){**

09

10 /\*tService.trace\_open({

11 ktbs\_base: "http://your.server.name/ktbs/your\_base/",

12 name: "trc\_demo",

13 success: function(){console.log("success is callbacked");},

14 error: function(jqXHR,textStatus, errorThrown){console.log("error is callbacked.");}

15 });\*/

16 ***var*** begin **=** end **=** **(*new*** Date**()).**getTime**();**

17 tService.trace\_put\_obsels**({**

18 trace\_uri**:** "http://your.server.name/ktbs/your\_base/trc\_demo/"**,**

19 model\_uri**:** " http://your.server.name/ktbs/your\_base/model\_demo/"**,**

20 obsel**:** **{**

21 type**:** "click"**,**

22 begin**:** begin**,**

23 end**:** end**,**

24 subject**:** "user\_demo"**,**

25 application**:** "app1"

26 **},**

27 success**:** ***function*(){**console.log**(**"success is callbacked"**);},**

28 error**:** ***function*(**jqXHR**,**textStatus**,** errorThrown**){**console.log**(**"error is callbacked."**);}**

29 **});**

30 **}**

Exemple : Exemple de code utilisant le package de collecte de traces modélisées.

## Authentification et Synchronisation session utilisateur

Le serveur d’assistance est une application web autonome mais il n’a pas d’authentification propre. C’est l’utilisateur de l’application qui fait appel à l’assistant qui doit être aussi l’utilisateur de l’assistance, et ceci de manière certaine et contrôlée. En effet, les traces d’interactions collectées ne doivent pouvoir être exploitées par l’assistant que pour le compte, ou avec la délégation du compte de l’utilisateur de l’application tracée. Pour faire cela, une cryptographie symétrique (clé secrète) et un protocole de synchronisation session sont utilisés. Il y a trois étapes dans le protocole d’authentification et et de synchronisation pour session utilisateur partagée application+assistance :

1. Partager une clé secrète
2. Créer une nouvelle session utilisateur
3. Synchronisation utilisateur

### Partager une clé secrète

Pour sécuriser les messages échangés entre l’application (serveur) et l’assistance (serveur), une clé secrète est partagée entre les deux parties au travers d’un protocole sécurisé (HTTPS), comme illustré dans la Figure 2.

Cette clé est unique pour chaque serveur application qui demande un service d’assistance. Le service d’assistance doit donc gérer autant de clés qu’il y a d’applications demandant son assistance. Le serveur assistance permet à plusieurs serveurs application de se connecter à lui. Donc, un ensemble de clés est géré par le distributeur de clés.



Figure 2. Partager la clé secrète

### Créer une nouvelle session utilisateur

A cette étape, si les applications permettent typiquement aux utilisateurs d’ouvrir une session web avec un système d’assistance basé sur un autre serveur, le système assistance ne peut pas encore identifier quel utilisateur le sollicite et quelle base de M-traces il peut exploiter à sa demande et de manière sécurisée. Pour permettre cette identification sécurisée, l’information de la session utilisateur en cours est encryptée et envoyée au serveur assistance selon le principe illustré dans le schéma de la Figure 3.



Figure 3. Créer une nouvelle session utilisateur

Avant l’ouverture d’une nouvelle interface assistance, une requête est envoyée au serveur application afin d’obtenir un ticket encrypté par la clé secrète créée à l’étape précédente et partagée (1) avec le serveur assistance. Ce ticket contient l’information sur la session utilisateur courant comme identifiant utilisateur, identifiant session, etc. (2)

Immédiatement après l’ouverture d’une nouvelle session, ce ticket est envoyé au serveur assistance pour le valider (3). Ensuite, le serveur assistance décrypte le ticket avec la clé secrète correspondant à l’application. Si le décryptage est un succès avec la clé secrète, l’identifiant utilisateur de l’application est utilisé comme identifiant utilisateur sur le serveur assistance (4) qui peut alors accéder comme l’application à la base de M-traces de l’utilisateur.

La date du ticket (Di) est ajoutée dans le message pour limiter la réutilisation de ticket ainsi que la possibilité de décryptage des attaqueurs (système de bail).

### Synchronisation utilisateur

Le schéma de la Figure 4 illustre cette problématique de synchronisation. Il faut en effet pouvoir suivre en temps réel les modification d’état de session application et en particulier les modifications de l’état utilisateur (i.e. changement d’utilisateur, connexion, déconnexion) (1). Pour assurer cette synchronisation, ces changements sont propagés automatiquement au serveur assistance pour assurer la cohérence des sessions utilisateur sur deux serveurs différents (2).



Figure 4. Synchronisation utilisateur

# Références API

## Trace Service (tService)

### trace\_open (options)

Cette fonction sert à créer une trace dans le système de gestion de base de M-traces.

Paramètre « options » (type: Object) est un ensemble de paires clé/valeur qui configurent la fonction.

* ktbs\_base : L’URI de la base de traces
* name : le nom de trace
* success : la fonction callback invoquée dans le cas de création de la trace est succès.
  + ret : le string de réponse de la base de données (dépend du SGBT cible)
* error : la fonction callback invoquée dans le cas de création de la trace échoue.
  + jqXHR : (jQuery XMLHttpRequest) remplace l’objet XMLHttpRequest native du navigateur
  + textStatus : un string décrit le type d’erreur qui se produit
  + errorThrown : un objet exceptionnel s’il se produit
* async (defaut=true) : Par défaut, toutes les requêtes sont envoyées de manière asynchrone. Si vous avez besoin requêtes synchrones, définir cette option à false.

### trace\_put\_obsels (options)

Cette fonction sert à insérer les obsels dans une trace donnée.

Paramètre « options » (type: Object) est un ensemble de paires clé/valeur qui configurent la fonction.

* trace\_uri : le chemin d’accès de la trace
* model\_uri : le chemin d’accès du modèle des obsels insérés
* obsels : un ensemble d’objets représentant des obsels qui vont être mis dans la trace.
* success : la fonction callback qui s’est invoqué dans le cas la création de la trace est succès.
  + ret : le string de réponse de la base de données
* error : la fonction callback qui s’est invoqué dans le cas la création de la trace n’est pas réussie.
  + jqXHR : (jQuery XMLHttpRequest) remplace l’objet XMLHttpRequest native du navigateur
  + textStatus : un string décrit le type d’erreur qui se produit
  + errorThrown : un objet exceptionnel s’il se produit
* async (defaut=true) : Par défaut, toutes les requêtes sont envoyées de manière asynchrone. Si vous avez besoin requêtes synchrones, définir cette option à false.

### trace\_get\_obsels(options)

Cette fonction sert à récupérer les obsels dans la trace.  
Paramètre « options » (type: Object) est un ensemble de paires clé/valeur qui configurent la fonction.

* trace\_uri : le chemin d’accès de la trace
* success : la fonction callback qui s’est invoqué dans le cas la récupération de la trace est succès.
  + obsels : l’ensemble des obsels récupérés.
* error : la fonction callback qui s’est invoqué dans le cas la création de la trace n’est pas réussie.
  + jqXHR : (jQuery XMLHttpRequest) remplace l’objet XMLHttpRequest native du navigateur
  + textStatus : un string décrit le type d’erreur qui se produit
  + errorThrown : un objet exceptionnel s’il se produit

## Assistance Service (tAssistance)

### obsel\_serialize\_html (options)

Cette fonction sert à sérialiser un obsel sous format HTML avec ses propriétés.  
Paramètre « options » (type: Object) est un ensemble de paires clé/valeur qui configurent la fonction.

* obsel (type : Object) : l’obsel va se sérialiser.

La *valeur de retour* est un string représentant l’obsel en HTML.

### draw\_obsels (options)

Cette fonction sert à créer une vue interactive des obsels.  
Paramètre « options » (type: Object) est un ensemble de paires clé/valeur qui configurent la fonction.

* parentNode (type : String ou DOMDocument): réfèrent à la position que la vue va être injectée
* obsels (type : ArrayObject) : un ensemble d’objets (obsels)
* getx (type : fonction) : la fonction récupère les valeurs représentant sur l’axe « x » de la vue.
* gety (type : fonction) : la fonction récupère les valeurs représentant sur l’axe « y » de la vue.
* width (Integer): la largeur de la vue
* height (Integer): la hauteur de la vue
* obselStyle : un script Javascript pour décorer l’obsel en SVG basé sur l’information de l’obsel

## Assistance Server

### User

#### Mettre à jour l’état de session utilisateur

Pour mettre à jour l’état de session utilisateur, créez une requête HTTP GET à l’URI du service assistance :

[/srv/](http://your.assistance-server.name/tAssistance/)service.php

**GET paramètres**

|  |  |  |
| --- | --- | --- |
| Paramètre | Valeur | Description |
| App\_id | Obligatoire | L’identifiant de l’application qui crée la requête |
| Session\_id | Obligatoire | L’identifiant de la session utilisateur sur le serveur application |
| User\_id | Obligatoire | L’identifiant de l’utilisateur en cours dans la session.  Le paramètre est 0 si l’utilisateur s’est déjà déconnecté. |

**Réponse**

Après l’envoi de la requête, l’application va recevoir une réponse HTTP.

|  |  |  |
| --- | --- | --- |
| Statut | Réponse/*Exemple* | Description |
| 200 | NULL | Le changement de l’état de session utilisateur est mis à jour |
| 500 | *Cannot connect to database* | Une erreur s’est produite |

#### Obtenir une clé sécrète

Pour obtenir une clé sécrète sur le serveur application, créez une requête HTTP GET à l’URI du service assistance :

[/srv/](http://your.assistance-server.name/tAssistance/)service.php

**GET paramètres**

|  |  |  |
| --- | --- | --- |
| Paramètre | Valeur | Description |
| App\_id | Obligatoire | L’identifiant de l’application qui crée la requête |

**Réponse**

|  |  |  |
| --- | --- | --- |
| Statut | Réponse/*Exemple* | Description |
| 200 | *fruoskfhd* | La clé en texte brut |
| 500 | *Cannot connect to database* | Une erreur s’est produite |

### Style

#### Créer un style

## Trace extension for application (client) : tApp/service.js

### Ouvrir une nouvelle fenêtre assistance

tApp.openAssistWindow() -> void

Description : ouvrir une fenêtre connectée au serveur assistance.

## Trace extension for application (server) : tApp/service.php

### Synchronisation session utilisateur

#### Obtenir une clé avec le serveur assistance

Pour obtenir la première clé sécrète avec le serveur assistance, créez une requête HTTP GET à l’URI de l’interface extension trace de l’application :

/tApp/service.php

**GET paramètres**

|  |  |  |
| --- | --- | --- |
| Paramètre | Valeur | Description |
| action | get\_private\_key | Obligatoire |

**Réponse**

|  |  |  |
| --- | --- | --- |
| Statut | Réponse/*Exemple* | Description |
| 200 | NULL | L’application obtient une nouvelle clé avec le serveur assistance |
| 500 | *Cannot connect to database…* | Une erreur s’est produite |

#### Obtenir un ticket valable sur le serveur assistance

Pour obtenir un ticket valable sur le serveur assistance, créez une requête HTTP POST à l’URI de l’interface extension trace de l’application :

/tApp/service.php

**POST paramètres**

|  |  |  |
| --- | --- | --- |
| Paramètre | Valeur | Description |
| action | get\_ticket | Obligatoire |

**Réponse**

|  |  |  |
| --- | --- | --- |
| Statut | Réponse/*Exemple* | Description |
| 200 | *H8RAqjwEepXVa7pzFd4jixFKDw6MlqZe6/oE+L0KFLRcpZlytQguuPEt+itzHXHnygd27QlTDHboWLCUW47Ot5HVxVejo9W0Ep60omQa3Wk=* | Le ticket encrypté par la clé sécrète entre l’application et l’assistance (le standard d’encryptage est AES-256-CBC). Voir http://www.openssl.org/docs/apps/enc.html#SUPPORTED\_CIPHERS |
| 500 | *Cannot connect to database* | Une erreur s’est produite |