TALTIONI APPLICATION DEVELOPER'S GUIDE

Authors: Medixine Oy

Editor	Date	Version	Comment
JE	19.10.2012	1.0	First accepted version without Draft label
VN	22.10.2012	1.01	OAuth Application Name term change, culture parameter added to OAuth request
JE	7.11.2012	1.02	Fixed code sample, changed example URLs and some terms
JE	21.11.2012	1.03	Cleaned up code samples, added more terms in terms & concepts
JE	29.11.2012	1.04	Added Java code samples, changed client_id to use OAuth Application Name in sample in chapter 4.6.
JE	14.12.2012	1.05	More information on data types and binary data handling (new chapter 5.2). Info on singleWsdl option.
JE	29.1.2013	1.06	Minor update, some clarifications to the OAuth process added and URLs changed to match the test environment.

TABLE OF CONTENTS

1. INTRODUCTION

This document is aimed for application developers who wish to implement a Taltioni-compliant application. The document provides detailed information and complete instructions for creating a Taltioni application that interfaces with Taltioni Authorization Server and Taltioni Web Services API.

Taltioni Authorization Server provides an OAuth 2.0 compliant endpoint for authorizing your application. Each application can only access the information of users who have performed the authorization procedure. OAuth is a well–adopted IETF standard that is used by major vendors, such as Google and Facebook.

After the authorization procedure, your application can access the user's health record via the Taltioni Web Services API. The Web Services API provides a WSDL file and SOAP endpoints. All data contracts are strongly-typed and self-descriptive. The API is described in detail in document *Taltioni Web Services API Reference*.

All Authorization Server and Web Services API interfaces are based on globally accepted standards and a lot of tooling is available for all major development platforms. The document contains a list of popular development tools for different platforms.

All Taltioni applications must pass an audit procedure before they will be able to access any Taltioni services. This audit procedure must be agreed separately with Taltioni Cooperative and is outside the scope of this document.

2. TERMS AND CONCEPTS

2.1. Taltioni terminology

The table below lists Taltioni-specific terms used in this document:

Name	Description
Health Record	Health record of a single citizen, contains personal information and health record items, such as medications and observations.
Core Health Record Type	Core health record types are pre-defined static types exposed by the Web Services interfaces. Core health record types are: Allergy, Condition, Immunization, HealthRecordFile, LabTestResult, Medication and Procedure.
Observation Type	An observation represents a single observation related to a person, such as a weight or blood glucose reading. Observation types are dynamic; new types can be created without changes to the Web Services interfaces.
Taltioni User	A citizen who has registered as a Taltioni user. Each user can own several health records. A user can also be authorized to access health records created by other people. Taltioni applications cannot access user identifiers or other user information directly. All operations are performed within the context of a single health record.
Taltioni Application Registry	Registry of all active Taltioni applications that have passed the audit procedure

Taltioni Application	Taltioni-compliant application registered in the Taltioni Application Registry
Taltioni Authorization Server	OAuth 2.0 compliant authorization provider
Taltioni Web Services API	Interfaces for accessing and managing health record information
Taltioni Web Services API Reference	Document that describes the Web Services API
Taltioni Data Types Reference	Document that describes the supported static and dynamic data types
Taltioni Authentication Server	Strong authentication provider for authenticating Taltioni users
OAuth Application Name	Human-readable unique name for the application, used in the OAuth authorization process
OAuth Redirect URL	URL in your application where the authorization response should be sent. Optional, can be set dynamically for each authorization request.
ApplicationId	Globally unique identifier for the application, used in Web Services calls
Application URL	Landing page of your application, used in the public application list of the Application Registry. Not used for any technical purposes.
Authorization Server Credentials	Username and password for accessing authorization services
Shared Secret	Secret code used for Web Services message authentication
Access Token	Access token needed to call the Taltioni Web Services API. Access Token identifies both the health record and the user and is only valid for a specific application-health record-user combination. The user and health record may not point to the same person because a single user may have access to several health records.
Authorization Server Endpoints	OAuth 2.0 -compliant endpoint URLs
Web Services API Endpoint	WSDL file URL for the API

2.2. General technology terminology

The table below lists general technology terms used in this document:

Name	Description

WSDL	Web Services Description Language, W3C standard that describes services as a set of endpoints that operate on messages
SOAP	Simple Object Access Protocol, W3C standard that provides support for exchanging structured information in Web Services implementation
MTOM	Message Transmission Optimization Mechanism, W3C standard that enables efficient sending of binary data to and from Web Services
HTTPS	Hypertext Transfer Protocol Secure, encrypted communication protocol used for all communication between a Taltioni application and Taltioni servers
OAuth 2.0	IETF standard for 3rd party application authorization
OAuth Resource Owner	OAuth term that corresponds to a Taltioni user
OAuth Client	OAuth term that corresponds to a Taltioni application
OAuth Resource Server	OAuth term that corresponds to Taltioni Authorization Server
OAuth Authentication Provider	OAuth term that corresponds to Taltioni Authentication Server
JSON	JavaScript Object Notation, lightweight data-interchange format used in OAuth 2.0

3. BASIC STEPS

Each application has to follow these steps to access Taltioni services.

3.1. Register your application

Contact Taltioni Cooperative and arrange an audit for your application.

To be listed in the *Taltioni Application Registry* you need to provide the following information about your application:

- **Vendor**
- **Application title**
- **Application description**
- **Application URL**
- **Application image (size and format to be defined)**
- **OAuth Redirect URL**
- **Required API operations and data types**

During the audit you will be asked for more information, but this is the minimum information set needed for including the application in the registry.

See document *Taltioni Web Services API Reference* for information on the API operations and document *Taltioni Data Types Reference* for information on the supported data types.

After a successful audit the application is registered in the Taltioni Application Registry and you will receive the following information:

- **ApplicationId**
- **OAuth Application Name**
- **Authorization Server Credentials**
- **Shared Secret**
- **Authorization Server Endpoints**
- **Web Services API Endpoint**

OAuth Application Name, ApplicationId and Shared Secret are application-specific and must only be used with the audited application. ApplicationId is a GUID (globally unique identifier) that is used in Web Services API calls. OAuth Application Name is used as the client identifier in the OAuth authorization process.

All registered applications are listed on the Taltioni.fi web site.

3.2. Authorize your application

Before your application can access Taltioni health record information, the users must authorize your application. You will need the OAuth Application Name, Authorization Credentials and Authorization Server endpoint addresses during the authorization process.

A Taltioni user can own or have access to several health records, but the authorization procedure only grants access to a single health record chosen by the user.

At the end of the authorization procedure your application receives an Access Token that is needed to access the user's information via the Web Services API.

The user can revoke your application's access at any time. If access is revoked, the only way to access the user's information is to request the user to perform the authorization procedure again. In this case a new access token is issued and the old token becomes obsolete. Your application can also inactivate the link to the user via the Taltioni Web Services API.

See chapter 4 for detailed instructions on the application authorization procedure.

3.3. Access Taltioni Web Services API

When a user has authorized access to a health record, your application can call the Web Services API. You will need the ApplicationId, Shared Secret, Access Token and WSDL file URL to access the service. The shared secret is used to generate a message authentication code that is needed for all Web Service calls.

The WSDL file provides complete schemas for all data items exposed through the service. The data contracts are well-structured and strongly-typed, which makes the interface robust and self-descriptive. It also supports Web Services client generation tools.

See chapter 5 for detailed instructions on accessing the Taltioni Web Services API.

4. APPLICATION AUTHORIZATION PROCEDURE

You will need the following information during the authorization procedure:

- **OAuth Application Name:** Identifies your application
- **Authorization URI:** Endpoint for starting the authorization procedure
- **Token URI:** Endpoint for getting the Access Token
- **Authorization Server Credentials:** Credentials needed to access the endpoints

Authorization procedure data flow is depicted in figure 1 (OAuth terms are shown in parentheses).

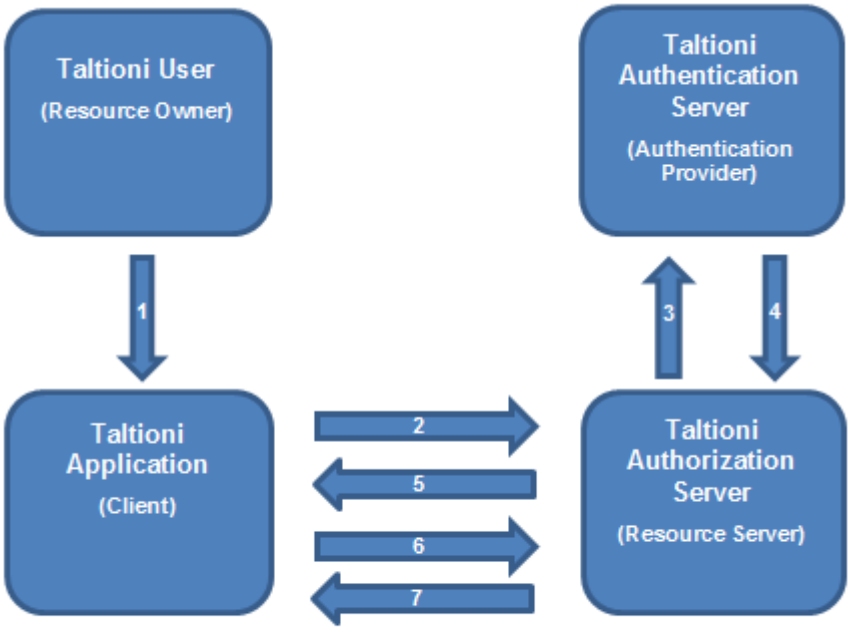

Figure 1: Application authorization data flow

Each phase is described in details in the next chapters.

4.1. User logs in the application (1)

The user must first authenticate in your application. Your application can use any authentication method as longs as the method has been accepted in the audit phase.

4.2. Application asks for authorization from Authorization Server (2)

After the user has been authenticated, you can present the user with a link for authorizing your application.

Build the authorization request URI with the following parameters:

- **response_type**

  REQUIRED. Static value, MUST be set to "code".

- **client_id**

  REQUIRED. Your OAuth Application Name.

- **redirect_uri**

  OPTIONAL. URI in your application where the authorization response should be sent. The static OAuth Redirect URL of your application is used if not provided.

state

  OPTIONAL. Optional value for maintaining state information between the request and callback. Normally this is not needed, but your application can choose to use it for its own purposes.

- **culture**

  OPTIONAL. Optional value for culture used in OAuth process. Use format <lang>-<locale>, e.g. fi-FI or sv-FI (two-letter ISO format for language and locale)

The request must be sent to the Authorization URI provided to you after the audit.

The network protocol is HTTPS and the request method is GET. The request must be made by the user's browser and the authentication cookie must be present in the request.

All parameters must be in "application/x-www-form-urlencoded" format. See http://www.w3schools.com/tags/ref_urlencode.asp for more information.

Taltioni Authorization Server endpoints are secure HTTPS endpoints and the server certificates are verified by a globally trusted certification authority.

Example:

- **Authorization URI:** asiakastesti.taltioni.fi/Authorize (use the actual Authorization URI provided)
- **response_type:** code
- **client_id:** MyTaltioniApp
- **redirect_url:** https://myapp.com/AuthCallback

Resulting URI:

https://asiakastesti.taltioni.fi/Authorize?response_type=code&client_id=MyTaltioniApp&redirect_uri=https%3A%2F%2Fmyapp.com%2FAuthCallback

4.3. User is redirected to Taltioni Authentication Server (3)

This phase does not require actions from your application, but is included for clarity.

The user is redirected to the Taltioni Authentication Server for authentication and identification.

Note that if your application uses Taltioni Authentication Server for authentication, single sign-on can be performed and the user is not required to provide authentication credentials.

4.4. Authentication Server returns authentication result (4)

This phase does not require actions from your application, but is included for clarity.

If the user was successfully authenticated, the Authentication Server returns the user's personal identifier to the Authorization Server. The user is then logged in Taltioni Authorization Server.

The user selects which health record your application is linked to and is shown a page containing information about your application. The data operations and data types required by your application are also shown to the user. The user can then grant or deny access.

4.5. Authorization Server returns authorization result (5)

4.5.1. Authorization success

If the authorization was successful, the Authorization Server creates a unique Authorization Code and sends the following response to the URI indicated by redirect_uri in the authorization request:

- **code**
  The authorization code created by the Authorization Server. The code will expire in ten minutes if it is not used. The code can be used only once and is bound to the client_id and redirect_uri. The length of the code is 32 characters.
- **state**
  Optional value, included if provided in the authorization request.
  Your application must ignore all other parameters.
  The response is sent as a GET HTTP(S) request.

Example response URI:

https://myapp.com/AuthCallback?code=00b938b6e10e4c1d89083be5ec58febc

4.5.2. Authorization error

If the user cancels the authentication procedure in Taltioni or the Taltioni Authentication Server doesn't send a response back to the Authorization Server, no response is sent to your application.

If the authorization request fails because of a missing, invalid or mismatching redirection URI or missing or unknown client identifier (OAuth Application Name), an error page is shown to the user, but no response is sent to your application.

If the user denies the access or the request fails for other reasons not listed above, the Authorization Server sends the following response to the redirection URI:

- **error** That can be either:
  - invalid_request

The request is missing a required parameter, has invalid parameter value, includes a parameter more than once, or is otherwise malformed.

- unauthorized_client

  Your application is not authorized to request authorization code using this method.

- access_denied

  The user or Authorization Server denied the request.

- unsupported_response_type

  The requested authorization response type is not supported.

- server_error

  The Authorization Server encountered an unexpected error.

- temporiraly_unavailable

  The Authorization Server is not available right now due maintenance or overload.

- **error_description**

  UTF-8 encoded text providing additional information about the problem.

- **state**

  Value provided by the client if it was included in the authorization request.

The response is sent as a GET HTTP(S) request.

Example error response URI:

https://myapp.com/AuthCallback/?error=invalid_request

4.6. Application requests Access Token (6)

The authorization result is a temporary code that is used to get the final Access Token needed to access the user's health record.

Build the access token request URI with the following parameters:

- **grant_type**

  REQUIRED. Value must be set to "authorization_code".

- **code**

  REQUIRED. The authorization code received from the resource server.

- **redirect_uri**

  REQUIRED, if it was used in the authorization request. It must be identical to the value in the authorization request.

- **client_id**

  REQUIRED. Your OAuth Application Name.

The request must be sent as an HTTPS POST request to the Token URI provided to you. The above parameters must be included in the posted data.

The request must also include a Basic Authentication authorization header based on the Authorization Server Credentials (username and password) provided to you. The header value

must follow syntax "Basic <value>", where <value> is a Base64 encoded string of the concatenated username and password with a colon in between. The input string (username:password) must be in UTF-8 format.

For example, if the username is "Aladdin" and password "open sesame", the string to be Base64 encoded would be "Aladdin:open sesame" and would result in value "QWxhZGRpbjpvcGVuIHNlc2FtZQ==".

Important: the token request must be sent directly from your application server, not from the user's browser! The request must not include any cookies or other information that is not relevant to the OAuth process. The information sent in the token request is confidential and must never be accessible by the user or the user's browser. Applications that do not respect this rule will not pass the audit process.

Example:

URL: https://asiakastesti.taltioni.fi/RequestToken

HTTP header:

```
POST /RequestToken HTTP/1.1
Host: asiakastesti.taltioni.fi
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
Content-Type: application/x-www-form-urlencoded;charset=UTF-8
```

HTTP body:

```
grant_type=authorization_code&code=00b938b6e10e4c1d89083be5ec58febc&client_id=MyTaltioniApp
```

4.7. Authorization Server returns access token (7)

4.7.1. Access Token success

If the access token request was authorized, the Authorization Server creates a unique Access Token and sends the following response with HTTP 200 (OK) status code:

- **access_token**

  The access token issued by resource server. Token length is 32 characters.

- **token_type**

  Static value set to "taltioni_token".

The parameters are sent in the HTTP response body using the "application/json" content type. Parameter names and string values are JSON strings. Numerical values are JSON numbers. The order of the parameters does not matter.

Example success response:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "access_token":"33369431943e4fadb2629bb66a8dafa4",
```

```
 "token_type":"taltioni_token"
}
```

The client MUST ignore unrecognized value names in the response.

See chapter 6.1 for a list of JSON tools for different development platforms.

4.7.2. Access token error

If the request fails or is invalid, the Authorization Server sends the following response with HTTP 400 (Bad request) status code:

- **error** A single error code from the following:
  - invalid_request

    The request is missing a required parameter, includes an unsupported parameter value (other than grant type), repeats a parameter, includes multiple credentials, utilizes more than one mechanism for authenticating the client, or is otherwise malformed.
  - invalid_client

    Client (i.e. application) authentication failed (e.g. unknown client, no client authentication included, or unsupported authentication method).
  - invalid_grant

    Authorization code is invalid, expired, revoked, or does not match the redirection URI used in the authorization request, or was issued to another client.
- **error_description**

  UTF-8 coded text providing additional information about the problem.

Parameters are sent in HTTP response body using the "application/json" content type. Parameter names and string values are JSON strings. Numerical values are JSON numbers. The order of parameters does not matter.

Example error response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "error":"invalid_request"
}
```

5. Accessing Taltioni Web Services API

Taltioni Web Services API is a standards-compliant Web Service that exposes a WSDL file and SOAP endpoints. HTTPS is used for all communication and MTOM is used for optimizing binary data transfer.

The interfaces provided by the Web Services API are synchronous, but most Web Services frameworks support the generation of asynchronous client proxies. However, the operations have been designed to be fast so asynchronous operations are not required unless you are looking for very high scalability.

Test environment WSDL URL:

https://asiakastestipalvelut.taltioni.fi/Taltioni

If you encounter problems with client proxy generation or wish to get the WSDL as a single file without references to separate schema files, you can call URL:

https://asiakastestipalvelut.taltioni.fi/Taltioni?singleWsdl

5.1. Operations

There are two types of operations in the API:

- **General operations:** Get information on the supported data types and codes; Access Token is not required
- **Health record operations:** Get or set information related to a specific health record; Access Token is required

Operations supported in Taltioni Web Services API 1.0:

Operation	Description	Type
About	Returns basic information about the service. The operation can be used to test basic connectivity. It can be called without any headers or tokens.	General
GetCodes	Gets codes that are used to specify data types, units and other codable values.	General
GetHealthRecordItemTypes	Gets detailed metadata on the supported health record item types.	General
GetApplicationInfo	Returns information about your application, including the access rights given to your application.	General
CreateHealthRecordProfile	Creates a new health record profile (restricted access, only available for specific applications)	General
GetAuthorizationInfo	Returns information about the link between the user and your application.	Health record
GetHealthRecordItems	Gets health record items according to the given criteria. Note that this operation does not return any binary data.	Health record
GetHealthRecordItemsPaged	Gets a specified number of health record items according to the given criteria. Can be used for paging item lists. Note that this operation does not return any binary data.	Health record
GetHealthRecordItem	Gets health record item with the given type and id. If the internal Taltioni id of the item is unknown, an alternative way is to supply fields SourceSystem and IdInSourceSystem. In this case field Id can be left empty. This is the only operation that returns binary data, i.e. HealthRecordFile.Data binary property is populated. Configure your service to use MTOM for optimal binary data transfer.	Health record

GetLatestHealthRecordItem	Gets the latest health record item of the given type.	Health record
StoreHealthRecordItems	Stores health record items. Update will be performed if an item with the given Id is found, otherwise a new item is created. Basic instructions: * All datetimes must be UTC * Leave Id empty for new items, assign correct Id for update operations * Important: when updating, populate the item with one of the GetHealthRecordItem-methods and only update the necessary fields! * The response contains a status (success/failed) for each item included in the request	Health record
DeleteHealthRecordItems	Deletes health record items. Supply the type and id of each item to delete.	Health record
GetHealthRecordProfile	Gets health record profile information. Note that some of the profile fields may not be available for your application.	Health record
UpdateHealthRecordProfile	Updates health record profile properties. Important: populate the profile with GetHealthRecordProfile and only update the necessary fields!	Health record
CheckPersonalIdentifier	Checks if the given personal identifier (HETU) is the same as the identifier stored in the Taltioni health record profile. The operation can be used to check that the AccessToken targets the correct person.	Health record
GetUserProfile	Gets the user profile. The user profile can be the same as the health record profile, or the user can be another person that has been authorized to access the health record.	Health record
RemoveApplicationAuthorization	Cancels the authorization, i.e. inactivates the link between your application and the user. Note that the user can also cancel the authorization at any time.	Health record

Note that all operations are not available for all applications. If you try to access an unauthorized operation, the service will return an access denied response (SOAP fault).

See document Taltioni Web Services API Reference for up-to-date descriptions of the supported operations.

5.2. Data types

Core data types Allergy, Condition, Immunization, HealthRecordFile, LabTestResult, Medication and Procedure have fixed schemas and data models that are contained in the schema files referenced by the WSDL file. The data model of these types cannot be changed without changing the WSDL file.

Observation types are dynamic types that have a fixed schema, but the data model is dynamic. This means that new observation types can be added to the service without changes to the WSDL file. Examples of observation types are Weight, Height and BloodPressure.

Each observation type contains one or more **observation item types**. Type Height has only one observation item type (Height), but type BloodPressure has several item types (Systolic,

Diastolic, Pulse, IrregularHeartBeatDetected, etc.). Each observation (observed data value, such a weight measurement) must contain an observation item for each mandatory observation item type.

Core data type **HealthRecordFile** must be used for storing any text or binary files. Images, PDF documents and other binary data cannot be stored using any other type. XML files or other text files, such as HL7 XML documents, must also be stored using the HealthRecordFile type. You must define the correct content type (i.e. mime type) for each file or they cannot be opened correctly by other applications. Common content/mime types are "application/pdf" (pdf document), "image/jpeg" (jpg/jpeg image), "image/gif" (gif image), "text/xml" (xml document) and "text/plain" (text document). For a comprehensive list of mime types, see http://www.iana.org/assignments/media-types/index.html. Supply the contents of the file in field *Data* (byte array that will be sent as a Base64 encoded string). ***If you get or store binary files, make sure you have enabled MTOM, which increases the performance for sending and receiving binary data*.**

See document Taltioni Web Services API Reference for up-to-date and detailed schemas of the data types. In the API reference, column *Occurs* indicates whether a field is required or not (Occurs value 1..1 indicates Required, value 0..1 indicates Optional). Column *Nillable?* only indicates if null is a valid value for the underlying data type; it cannot be used to check if a field is required.

Document *Taltioni Data Model* provides descriptions of all activated dynamic data types and information on the code sets related to each type.

Note that all item types are not available for all applications. If you try to access an unauthorized item type, the service will return an access denied response (SOAP fault).

5.3. SOAP request headers

All request messages must contain the following SOAP headers:

- **RequestId:** Unique id (GUID) for the request, must be generated by the client
- **Timestamp:** Current date and time in UTC
- **ApplicationId:** Globally unique application identifier
- **AuthCode:** Authentication code based on the header values and *Shared Secret*

Timestamp must be in UTC (Universal Time) in the format defined in http://www.w3.org/TR/xmlschema-2/#dateTime.

Requests that concern health record information contain an additional SOAP header:

- **AccessToken:** An access token received after a successful authorization procedure

5.3.1. Authentication code generation

Follow these steps to produce the AuthCode header value:

1. Concatenate RequestId, Timestamp, ApplicationId, AccessToken (if present in the request) and Shared Secret in this order, separated by a semicolon
2. Create an SHA-256 message digest of the concatenated string

The concatenated string must be encoded as UTF-8 before hashing.

Note that AccessToken must not be included in the AuthCode for general operations that do not target a specific health record (e.g. GetCodes).

Clock synchronization is not performed between the client and server so some discrepancy is allowed for the Timestamp, but you should use a clock synchronization server to ensure successful transactions.

Example:

RequestId: 936DA01F-9ABD-4d9d-80C7-02AF85C822A8

Timestamp: 2013-01-01T17:00:00Z

ApplicationId: 4007af84bc0f46f181d907e50f9f5a3a

AccessToken: 33369431943e4fadb2629bb66a8dafa4

Shared Secret: GfKq83HjkL90f94H (example value, use the Shared Secret provided to you after a successful audit)

Value to hash:

```
936DA01F-9ABD-4d9d-80C7-02AF85C822A8;2013-01-01T17:00:00Z;
4007af84bc0f46f181d907e50f9f5a3a;33369431943e4fadb2629bb66a8dafa4;GfK
q83HjkL90f94H
```

The resulting SHA-256 message digest must be sent as a byte array that will be converted to a Base64 encoded string during transport. Resulting digest in Base64 format:

```
gSOZttxv2BBQJmuU3FZ5ZJl7g7n6o/YJ4jtBPiCUF4E=
```

5.4. SOAP response headers

All response messages contain the following SOAP headers:

- **RequestId:** The request id sent in the request
- **Timestamp:** The timestamp sent in the request

5.5. Client proxy generation

It is strongly encouraged to use a Web Services framework that supports SOAP client proxy generation based on a WSDL file. Manual generation of the messages is time-consuming and error-prone.

See chapter 6.1 for a list of tools available for different development platforms.

5.6. Message examples

The message examples in this chapter are for demonstration purposes only and should not be used as reference for the actual implementation. See the Web Services API documentation and WSDL file for the current version of the schemas.

5.6.1. GetLatestHealthRecordItemRequest

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header>
    <Action s:mustUnderstand="1"
xmlns="http://schemas.microsoft.com/ws/2005/05/addressing/none">Taltioni.Services/T
altioniAPI/Actions/GetLatestHealthRecordItem</Action>
    <h:AccessToken
xmlns:h="http://taltioniapi.1.0.taltioni.fi">ac12804c53b145f5a128840631997cb4</h:Ac
cessToken>
    <h:ApplicationId
xmlns:h="http://taltioniapi.1.0.taltioni.fi">c820571a3f754560974a4e8432490a4e</h:Ap
plicationId>
    <h:AuthCode xmlns:h="http://taltioniapi.1.0.taltioni.fi">Base64 encoded byte
array</h:AuthCode>
    <h:RequestId xmlns:h="http://taltioniapi.1.0.taltioni.fi">15dad69c-ad0f-4a1d-
a767-6df82952f38b</h:RequestId>
    <h:Timestamp xmlns:h="http://taltioniapi.1.0.taltioni.fi">2012-08-
24T12:30:37.366077Z</h:Timestamp>
  </s:Header>
  <s:Body>
    <GetLatestHealthRecordItemRequest xmlns="http://taltioniapi.1.0.taltioni.fi">
      <ItemType>Allergy</ItemType>
    </GetLatestHealthRecordItemRequest>
  </s:Body>
</s:Envelope>
```

5.6.2. GetLatestHealthRecordItemResponse

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header>
    <h:RequestId xmlns:h="http://taltioniapi.1.0.taltioni.fi">15dad69c-ad0f-4a1d-
a767-6df82952f38b</h:RequestId>
    <h:Timestamp xmlns:h="http://taltioniapi.1.0.taltioni.fi">2012-08-
24T12:30:37.366077Z</h:Timestamp>
  </s:Header>
  <s:Body>
    <GetHealthRecordItemResponse xmlns="http://taltioniapi.1.0.taltioni.fi">
```

```
      <HealthRecordData xmlns:a="HealthRecordClient.Data"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
        <a:Allergies>
          <a:Allergy>
            <a:ApplicationId>c820571a3f754560974a4e8432490a4e</a:ApplicationId>
            <a:TypeId>Allergy</a:TypeId>
            <a:Id>f8d2cbf309964a8d90c2103b7c54f32a</a:Id>
            <a:Version>919f5e16f58f432fa28714aa8dd3c24e</a:Version>
            <a:EffectiveDateTime>2012-08-23T11:47:00Z</a:EffectiveDateTime>
            <a:Private>false</a:Private>
            <a:AuditInfo>
              <a:Created>2012-08-23T11:44:26.627Z</a:Created>
              <a:LastUpdated>2012-08-23T11:47:57.243Z</a:LastUpdated>
              <a:CreatedBy>Not available</a:CreatedBy>
              <a:LastUpdatedBy>Not available</a:LastUpdatedBy>
              <a:PerformerType>Unknown</a:PerformerType>
            </a:AuditInfo>
            <a:Name>Strawberry allergy</a:Name>
          </a:Allergy>
        </a:Allergies>
      </HealthRecordData>
    </GetHealthRecordItemResponse>
  </s:Body>
</s:Envelope>
```

5.7. Example scenarios

5.7.1. Scenario 1: Get weight measurements of a person, restricted to measurements stored by your application

.NET (C#, WCF)

Step 1: Generate client proxy class (TaltioniAPIClient):

- Open Visual Studio command prompt and run command:

```
svcutil https://asiakastestipalvelut.taltioni.fi/Taltioni
```

- Copy the generated cs file and config file to your solution

Step 2: Generate AuthCode:

```
Guid requestId = Guid.NewGuid();
string timestamp = XmlConvert.ToString(DateTime.UtcNow,
XmlDateTimeSerializationMode.Utc);
SHA256CryptoServiceProvider hashAlgorithm = new SHA256CryptoServiceProvider();
string toHash = requestId.ToString() + ";" + timestamp + ";" + ApplicationId + ";" +
AccessToken + ";" + SharedSecret;
byte[] authCode =
hashAlgorithm.ComputeHash(System.Text.Encoding.UTF8.GetBytes(toHash));

// Base64 conversion shown for reference only. .NET performs the encoding
automatically, but manual encoding may be necessary in some development platforms
string authCodeBase64 = Convert.ToBase64String(authCode,
Base64FormattingOptions.None);
```

Step 3: Call GetHealthRecordItems:

```
bool filterByApplicationId = true;
DateTime? startDate = null;
DateTime? endDate = null;

using (var client = new TaltioniAPIClient())
{
    var data = client.GetHealthRecordItems(AccessToken, ApplicationId, authCode, ref
requestId, timestamp, startDate, endDate, new string[] { "Weight" },
filterByApplicationId);
    var observations = data.Observations;
```

```
}
```

Java (Axis2)

Step 1: Generate client proxy:

- Use Axis2 wsdl2java.bat to generate the client proxy
- Copy the generated items to your solution

Step 2: Generate AuthCode:

```
formatUTC = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss'Z'");
formatUTC.setTimeZone(TimeZone.getTimeZone("UTC"));
authCode = (AuthCode)getTestObject(AuthCode.class);
originalText = UUID.randomUUID() + ";"+ formatUTC.format(new Date()) + ";"+
ApplicationId + ";"+ AccessToken + ";"+ SharedSecret;
md = MessageDigest.getInstance("SHA-256");
md.update(originalText.getBytes("UTF-8"));
dataHandler = new DataHandler(new ByteArrayDataSource(md.digest(), "application/octet-
stream"));
authCode.setAuthCode(dataHandler);
```

Step 3: Call GetHealthRecordItems:

```
arrayOfstring.addString("Weight ");
request.setItemTypes(arrayOfstring);
response = client.GetHealthRecordItems(request, AccessToken, ApplicationId, authCode,
RequestId, Timestamp);
observations[] = response.getHealthRecordData().getObservations().getObservation();
```

5.7.2. Scenario 2: Store new weight measurement performed using an electronic scale

See the first example for instructions on generating the client proxy class and AuthCode.

Mandatory fields for observations:

- TypeId
- EffectiveDateTime

Mandatory fields observation items:

- TypeId
- Value (one of the Value fields must be populated, NumberValue for measurements)
- Unit (mandatory for measurements, must match the unit defined for the item type)

Call operation GetHealthRecordItemTypes to get a full list of supported observation types. The list includes detailed information on each type, including TypeId, Unit and value type information.

See the Web Services reference for more information, such as descriptions of the value types and instructions on the fields that must be populated for the different dynamic types (Number, Text, Choice, YesNo, CodableValue, StructuredMeasurement and DateTime).

.NET (C#, WCF)

```
var data = new HealthRecordData()
{
    Observations = new Observation[]
    {
        new Observation()
        {
            EffectiveDateTime = DateTime.UtcNow,
            TypeId = "Weight",
            Notes = "Measured with MyScale",
```

```
            AuditInfo = new AuditInfo()
            {
                PerformerTypeCode = new HealthRecordClient.Data.CodedValue()
                {
                    Family = "Taltioni",
                    VocabularyName = "PerformerType",
                    Value = "Self"
                },
                SourceSystem = "Master data source",
                IdInSourceSystem = "123545737854574",
                DeviceInformation = new DeviceInformation()
                {
                    Manufacturer = "Superb Health Devices Inc.",
                    Model = "MyScale 1.0",
                    SerialNumber = "123456789",
                }
            },
            ObservationItems = new ObservationItem[]
            {
                new ObservationItem()
                {
                    TypeId = "Weight",
                    NumberValue = 80.5,
                    Unit = "kg"
                }
            }
        }
    }
};

bool abortOnError = false; // Do not abort even if some items cannot be stored

using (var client = new TaltioniAPIClient())
{
    var response = client.StoreHealthRecordItems(AccessToken, ApplicationId, authCode,
        ref requestId, timestamp, data, abortOnError);

    if (response.IsErrors)
    {
        foreach (var item in response.DetailedResults)
        {
            if (item.Success == false)
            {
                // Handle error for each unsuccessful item
            }
        }
    }
}
```

Java (Axis2)

```
observation.setApplicationId(ApplicationId);
observation.setEffectiveDateTime(Calendar.getInstance());
observationItem.setTypeId("Weight");
observationItem.setNumberValue(80.5);
observationItem.setUnit("kg");
arrayOfObservationItem.addObservationItem(observationItem);
observation.setObservationItems(arrayOfObservationItem);
arrayOfObservation.addObservation(observation);
healthRecordData.setObservations(arrayOfObservation);
request.setHealthRecordData(healthRecordData);
response = client.StoreHealthRecordItems(request, AccessToken, ApplicationId,
authCode, RequestId, Timestamp);
result = response.getStoreItemsResult();

if (result.getIsErrors()) {
    storeItemResults[] = result.getDetailedResults().getStoreItemResult();

    for (StoreItemResult item : storeItemResults) {
        if (!item.getSuccess) {
        // handle error
        }
    }
}
```

6. DEVELOPMENT TOOLS AND LINKS

6.1. Development tools

Use of existing libraries and tools for development is encouraged. It will make developing a Taltioni-compliant application easier and produce secure and reliable end results.

List of popular development tools that can be leveraged for Taltioni application development:

Platform	Tool	Description	URL
.NET	DotNetOpenAuth	Open source OAuth client library	http://www.dotnetopenauth.net/
.NET	WCF	.NET Web Services framework	http://msdn.microsoft.com/en-us/netframework/aa663324.aspx
.NET	Svcutil	.NET WS client proxy generator	http://msdn.microsoft.com/en-us/library/aa751905.aspx
.NET	JavaScriptSerializer	.NET JSON serializer	http://msdn.microsoft.com/en-us/library/system.web.script.serialization.javascriptserializer.aspx
.NET	SHA256CryptoServiceProvider	.NET SHA-256 provider	http://msdn.microsoft.com/en-us/library/system.security.cryptography.sha256cryptoserviceprovider.aspx
Java	Google OAuth Client Library	Google's OAuth client library	http://code.google.com/p/google-oauth-java-client/wiki/OAuth2
Java	Apache Amber	Open source OAuth client library	https://cwiki.apache.org/confluence/display/AMBER/Index
Java	Apache Axis2	Comprehensive Web Services framework	http://axis.apache.org/axis2/java/core/
Java	Axis2 wsdl2java	Axis2 WS client proxy generator	http://axis.apache.org/axis2/java/core/docs/userguide-creatingclients.html
Java	Axis2 MTOM	Axis2 MTOM support for WS client	http://axis.apache.org/axis2/java/core/docs/mtom-guide.html
Java	Axis2 JSON	Axis2 JSON support	http://axis.apache.org/axis2/java/core/docs/json_support.html
PHP	OAuth	PHP OAuth library	http://php.net/manual/en/book.oauth.php
PHP	WSO2	Comprehensive Web Services framework, supports MTOM	http://wso2.com/products/web-services-framework/php
PHP	SoapClient	PHP WS client proxy generator	http://www.php.net/manual/en/class.soapclient.php

PHP	jsonencode / jsondecode	PHP JSON serializer	http://www.php.net/manual/en/ref.json.php

6.2. Links

Useful links that provide more information on the technologies and standards relevant for Taltioni application development:

Description	URL
WSDL specification	http://www.w3.org/TR/wsdl
SOAP specifications	http://www.w3.org/TR/soap/
MTOM specification	http://www.w3.org/TR/soap12-mtom/
OAuth 2.0 specification	http://tools.ietf.org/html/rfc6749
OAuth 2.0 threat model and security considerations	http://tools.ietf.org/html/rfc6819
Google API authentication and authorization (uses OAuth2)	https://developers.google.com/accounts/docs/OAuth2
Facebook API authentication and authorization (uses OAuth2)	http://developers.facebook.com/docs/authentication/
JSON documentation	http://www.json.org/