Partners:

Join-In

Senior Citizens Overcoming Barriers by Joining Fun Activities

AAL Joint Programme: Project No. 031121

Deliverable: 4.2

Design and Implementation of the Join-In Platform

Date of deliverable: 31-10-12 / Version: 1.0

Lead contractor for this deliverable:
Norut

Contributors:
PAS, HMGU, VAL, ITC, NST

Dissemination Level:
Public

Project Duration: Nov. 2010 – Feb. 2014

Project co-founded by

Disclaimer

When citing the deliverable please use the following information

Join-In Project Report D<Nr> <Title>, <Year>. Retrieved <The date on which you retrieved the information> from <URL>.

A short note on the publication to the Coordinator (contact@join-in-for-all.eu) would be very much appreciated.

Table of Content

1 About Join-In

Join-In aims at providing the methodology and the technologies for elderly persons to participate in social activities and have fun via digital media. Loneliness in the elderly is a major problem in elderly care. Studies in Britain show that more than half of the people over the age of 75 live by themselves. Many of these suffer from loneliness and social isolation[1]. Activities offered by social services do, however, often not reach those most in need. Challenges for the elderly include: social deprivation, low self-esteem or physical inability. Social isolation and health are closely related and may lead to a variety of physical disorders and even depression. Studies have shown the correlation between loneliness and poor health. Especially the effects on immune system, the cardiovascular system and the onset of Alzheimer's disease could be shown[2][3][4].

The Join-In project aims at counteracting loneliness in the elderly by providing a concept, the methodology and technologies for elderly persons to participate in social activities.

[1] Office of National Statistics: Older people,. Living arrangements. At: http://www.statistics.gov.uk/cci/nugget.asp?id=1264

[2] CARMA – Care for the Aged at Risk of Marginalization (QLK6-CT-2002-03421) - Recommendations and Guidelines to Policy Makers. (2005). http://www.egga.ee/RecommendationsFinalwCoverTOC.pdf Last accessed:2/10

[3] Sorkin D, Rook KS, Lu JL: Loneliness, lack of emotional support, lack of companionship, and the likelihood of having a heart condition in an elderly sample. Ann Behav Med. 2002 Fall; 24(4):290-8

[4] Tomaka J, Thompson S, Palacios R: The relation of social isolation, loneliness, and social support to disease outcomes among the elderly. JAging Health.2006 Jun; 18(3):359-84

Fig.1 Join-In Platform

Join-In is setting up a social platform for the elderly; it allows communication by TV, Tablet and PC. A multi-player serious game for the elderly is being developed. The interest in gaming is high in seniors: In a survey performed in Germany with 1200 participants, age above 61, two out of three PC users stated that they enjoy playing games regularly on the internet[5]. Studies[6] could demonstrate the increase of cognitive skills, reaction times, self-esteem and the sense of well-being in the elderly when playing computer games. Another positive effect is that gaming is multigenerational and enables the elder generation socialising with the younger one, e.g. grandchildren. The concept includes exercising either by exergames or by moderated exercises as physical activity -besides supporting good health- counteracts the feeling of loneliness, while loneliness leads to less physical activity[7]. Recent results indicate that exergames create physical benefits and counteract loneliness[8.] Join-In encourages contacts with peers in the region and with family and friends living further afield - if necessary facilitated by an assistant.

Active participation is vital if the individual is to profit from the Join-In developments. Yet motivation for participation among the elderly is a challenge. One of the problems is the heterogeneity of the elderly, among other things regarding interests and health. Join-In is developing a methodology for elderly persons to participate in social activities. This is based on a thorough user requirement analysis. User groups are set up in Germany, Hungary, Ireland and Norway. The lead user group is based in Munich. Based on the results of the user requirement analysis and the analysis of relevant studies and related work a methodology for setting up a social networking platform which will encourage and enable involving homebound senior persons in social networking activities being developed. Digital inclusion and factors hampering its acceptance -such as accessibility, motivation, lack of skills and confidence- will be tackled and form part of the methodology. The involvement of user groups in four different countries will help us to achieve a European solution which will also be useful in other countries.

The Join-In project web-page:

http://www.join-in-for-all.eu

[5] OE24.at. Deutsche Studie - Sechs von zehn Senioren spielen am Computer. http://www.oe24.at/zeitung/digital/article318942.ece. Last accessed: 2/10

[6] Basak C, Boot WR, Voss MW, Kramer AF: Can training in a real-time strategy video game attenuate cognitive decline in older adults? Psychol Aging. 2008 Dec; 23(4): 765-77).OE24.at

[7] Hawkley LC, Thisted RA, Cacioppo JT: Loneliness predicts reduced physical activity: Cross-sectional & longitudinal analyses. Health Psychol. 2009 May; 28(3):354-63

[8] http://www.theatlantic.com/technology/archive/2011/02/physical-video-games-may-help-the-elderly-psychologically/71184

2 Introduction

In this report we present the design and implementation of the Join-In platform. However, since most of the effort has gone into the implementation of the platform and its components, the best description of the platform is to be found in the documented APIs and in the many lines of programming code on the Join-In source code account.

Fig shows the overall Join-in system with the components both at the user and system environment.

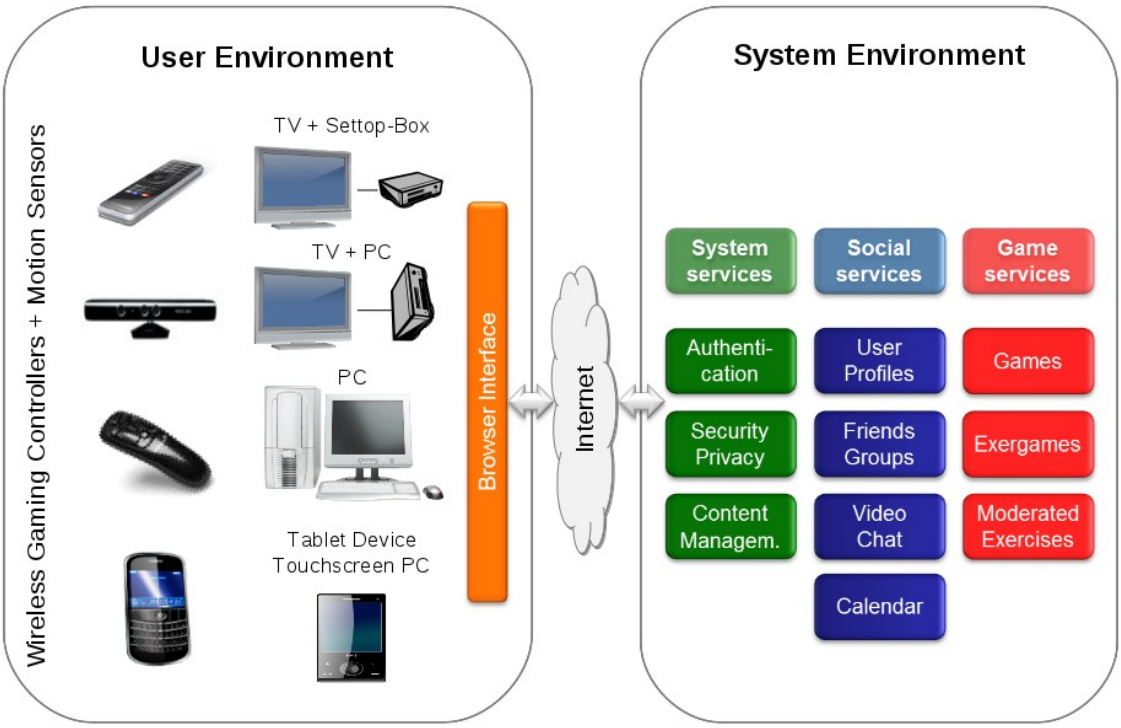


Fig.2 The Join-in user and system environment

The **user environment** is where the user accesses the Join-In services. The user environment will have various user devices; a hardware platform, and the software running on it, and game controllers and motion sensors. The **system environment** run the Join-In social network and portal and provides on-line services to the games and applications of Join-In.

In Join-In we have has designed and implemented a social network and social portal using the components of the Join-In platform. We have designed and implemented computer games and exercise games that can be accessed through the social network, and

developed special wireless controllers and interfaced commercial controllers and motion sensors to this platform.

Chapter 2 presents briefly the Join-In Connector, which provides an API for integration of the Join-In games, the social network portal, and existing social networks. Appendix A describes the detailed of the Join-In Connector API, while the source code is managed and shared on the Join-In source code account.

Chapter 3 presents briefly the Join-In social network server, and Chapter 3.1 presents the functionality supported and implemented on the portal. The server is based on ELGG. The source code of the functions is managed and shared on the Join-In source code account.

Chapter 4 presents briefly the user devices that will be supported by Join-In. They all run standard operating systems and browsers. Special attention is given to the Join-In controller.

Chapter 5 presents the architecture of the game servers, and describes the software components and drivers that have been developed by Join-In. The source code is managed and shared on the Join-In source code account.

3 Join-in social networking architecture

The social networking architecture of Join-In is according to Fig. 3 and depicts how the Join-In social network integrates the Join-In games, Join-In social portal and other applications, the , and how it can connect to existing social networks (e.g. Facebook, Google+, twitter).

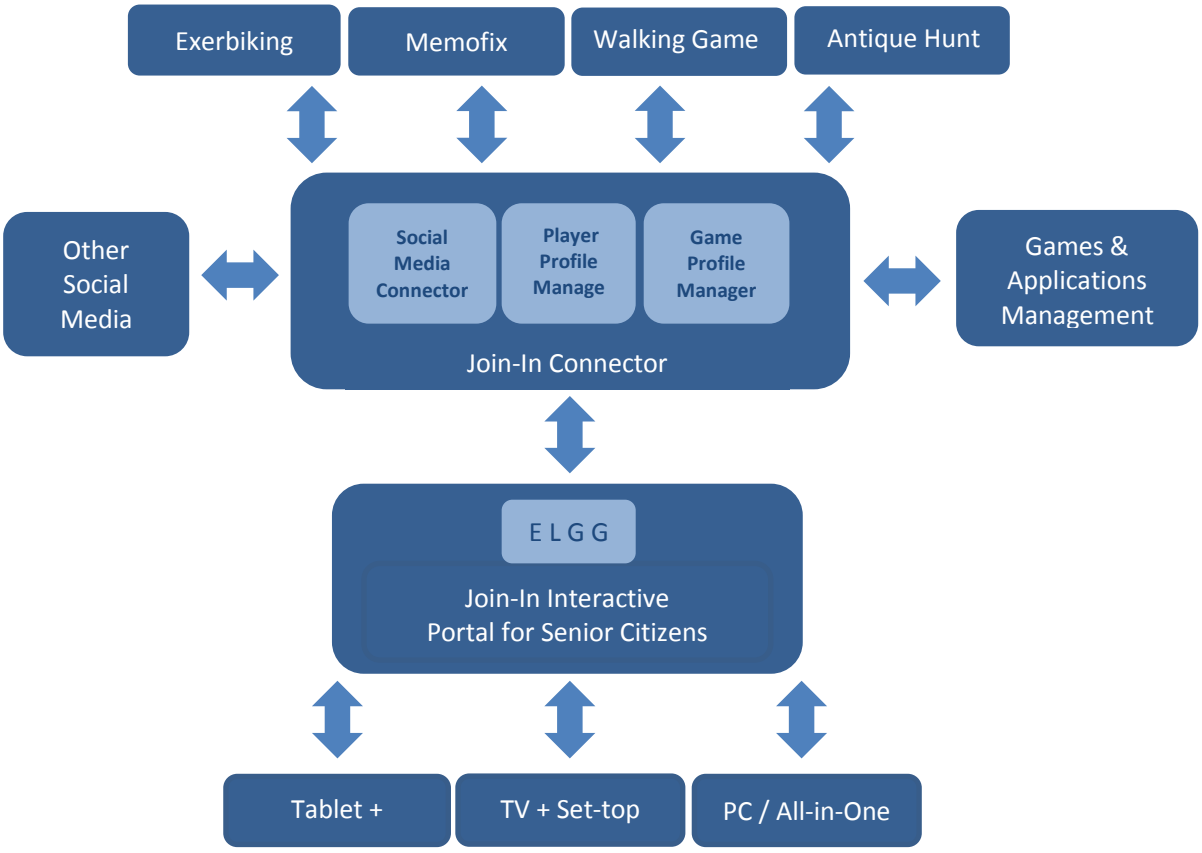


Fig.3 Join-In social networking architecture

The Join-In Connector contains:

- Game profile manager that will provide API for registering new games other applications.
- Player profiler manager that will provide an API to administrate the player's profile, for example adding profile, registering profile, listing profile etc.
- Social media connector that will enable the integration with other social media's and other AAL platforms.

The Join-In Connector is integrated with an ELGG platform which provides the portal.

The API of the Join-In Connector is described in further detail in Appendix A.

4 Social Network

In order to motivate and to socially connect elderly people to their friends, family and other people, the Join-In project builds a social network. The functionalities and content of the social network are defined based on requirements gathered from users groups and stakeholders.

The social network has to handle the different “Social Contact Functionalities” (e.g. user communication and profile setting), and to provide an easy way for adding or linking to the applications. These are

- Computer games
- Exercise games
- Exercising

At the same time, a graphical user interface that is suitable for the elderly users is required.

The social network has to take care of the special needs of the target group. This means

- Simple access for the elderly users as many have limited knowledge in new technologies
- Accessibility for users with physical impairments
- Multilingualism support for the users in the Join-In partner countries.

Based on user involvement we have identified a set of requirements for the social network, and the Join-In platform and social network portal will be adapted to the needs of the various user groups.

We have created 4 different roles that a user of the platform can have:

- System Administrator: The role of system administrator has full administrative rights on the platform. A system administrator controls the administration part and plug-ins and the usual responsibilities of a system administrator.
- User support: The role of user support covers several tasks: it allows you to manage the user registration and supports the users locally. In addition to the assistance by phone, a user supporter can also access the machine remotely - if the user has agreed to this option - and support him/her directly via remote access.
- Moderator: The moderator role allows inviting other users for exercises. It enables you to guide and moderate the different activities. Moderators may initiate an exercising session and connect users to jointly take part in an exercise session.

- User: The users have the right to change the default settings of their personal data and manage their own profile.

Having all this in mind, it was decided to use the ELGG framework for building the social network platform. ELGG is an Open Source and free project, based on a modular structure, allowing for the addition of new plugins on demand. Another reason for choosing this framework was the extensive library of add-ons and plugins already developed by the ELGG Community.

The Join-In social network support functions for

- Social contacts
- Tools for communicating and sharing
- Local and regional information
- Calendar and appointments
- Games and exergames
- Choosing avatars
- Exercising
- Help functionality
- Security and authentication issues

4.1 Functions

This section explains in more detail the different functionalities that the Join-In social network provides.

4.1.1 Social Contacts

User Profile

The User Profile contains the information that defines the user. It works within the social network as identification.

Additionally to the name, the User Profile includes:

- Photo
- Avatar
- Gamer profile

  - Skill level (per game)
  - High score (per game)
  - Favourite games
- Exercise profile
  - Favourite exercise
- Interests and hobbies
- Group memberships
- Restrictions (due to e.g. physical limitation)

All the information will be given on a voluntary basis.

User Settings

In addition to the User Profile - that captures the social profile - there are further details to be entered by the administrator when a new Join-In user is registered in the social network.

The User Settings include:

- First Name and Family Name
- User Name / Gamer tag
- User Id
- Contact details: Email and/or Telephone number

Friends

The user can add or delete friends and search new contacts.

Videoconferencing

Videoconferencing allows the users to have live video and audio chat as communication channel with their contacts.

The user will be able to invite his/her friends to a videoconference, making it more private.

It will allow the user to play a game while videoconferencing, adding a new social value to the game.

Text Chat

A chat plugin is provided. Some layout modifications in this ELGG community plugin are needed to improve the user's accessibility.

Messages

With the help of a keyboard (physical or virtual) the users will be able to send and receive written messages (similar to emails) to/from their contacts.

The messages optionally can be forwarded to an email account.

Exchange

The users are able to upload, share, search and comment, and tag different types of information, e.g.

- Photos
- Videos
- Web Links
- Texts: News, Histories, Poetry, etc…

The user may want to find in the history of files some precise content. For that purpose, a search engine is introduced and it will be used to index the content exchanged. The need of this functionality has to be evaluated by users.

Regional information

The regional information is a special set of user content: videos, stories and news from the local senior groups, stories from the neighbourhood, philosophy, theology, history, etc…. The major differences with the normal user content are:

- Open information: The information posted as regional information is accessible to all users or at least to all those users who are members of a defined region.
- Supervised: All the data posted as regional information is approved by a regional moderator to avoid misuse of the functionality.

This information helps the elderly to stay informed with the activities of their region, get in touch with new people, share photos, videos or opinions with their neighbourhood, etc...

4.1.2 Calendar and activities

Managing a user's appointments and schedule are functionality strongly related to users taking part in activities like games, exercises or even groups – thus helping the users to coordinate their social activities.

Calendar

The user can access a calendar to check personal appointments, events, and group or regional meetings.

Appointments and events

The user is able to add, delete or modify appointments or events within a graphical interface. This interface is a plugin for managing the users' calendar.

A specific date can be shared between friends or group members. That is helpful when, for example, an activity moderator wants to schedule that time and date for the next (exer)game or exercise. The moderator can set up an exercise goal, by adding a sequence of dates containing a set of exercise games and also video exercising. The users may also be able to see their progress in achieving the exercise goals in the calendar interface.

Reminders

A reminder is a special functionality that works directly with the calendar and helps the user to remember appointments and events.

The user will get an additional notification (e.g. by integrated messaging or email) for an upcoming appointment or event.

4.1.3 Games and Exergames

The games are an important part of the Join-In Platform and their integration with the Social Network is a major issue. The user can play the (exer)games with other users with or without the supervision of a Moderator, which will show how to play, observe the users movements and/or help and give feedback to the users.

Game registration

When a game has been developed, it has to be made available to the user. The social network is a really good tool for this purpose.

In order to offer game information to the users, there is a need of a plugin to implement the game information acquisition and of its integration in the social network. This plugin is the Game Registration. Once the game is registered, it will be accessible to the users in the game lists.

Games listing and gamer profile

There is a plugin to manage the lists of games offered and the list of games that the user already played. These lists are reachable by the Join-In Connector

Game launcher

It is necessary to provide easy access to the games. The game launcher gets information from the registered games on the game and gamer list.

Game metrics performance viewer

Both the cognitive games and exercise games (exergames) will collect performance data from the player over time. Each game that collects such data will allow the player to view their in-game progress for a defined period of time. This data will be readable by the Join-In Connector, so it can be stored and retrieved by the game servers. Read access may also be granted to other applications, e.g. for care providers who might have an interest in monitoring the performance and progress of the participants.

For each game the metrics list contains:

- total number of times the player have played a game
- total time spent on a game
- scores of single gaming sessions (including the high-score)
- total game score of a game
- level reached - if applicable
- goal to be reached: future appointments in the calendar

The nature of the data collected will vary between the different games, for example, the walking game may capture the following metrics per gaming session:

- Total number of steps taken

- Average number of steps taken per minute
- Number of obstacles avoided
- Time to complete the walking challenge

4.1.4 Avatar selection

An external application - similar to a game - enables the user to select and potentially even to modify the look of the avatar.

The avatar that the user chooses will be used in the (exercise) games the user plays, and possibly even when exercising. It can be used in the social network's user interface as an alternative to the user's photo.

4.1.5 Exercising

Join-In offers the users a variety of exergames for improving their physical fitness. In addition to these exergames, the elderly can participate in a remote gymnastic program, which can be with or without a moderator. For this purpose, the video sharing and the videoconferencing are helpful functionalities, as well as the calendar for organizing the exercise sessions.

4.1.6 Help

There will be a user manual consisting of several chapters.

One part describes the social platform, the way the social network is organised and how its features and tools are used.

An additional guideline explains step-by-step the course of action (for example: "enter your first name now"- click -using the mouse- the large blue button "Confirm" at the bottom of the screen). Thus the user can easily get acquainted with the platform. Screenshots and pictures will lead to a better understanding. This part will also be available as a printed manual.

Another part is directed at those persons that are regularly using digital media, e.g. grandchildren. It explains in detail all the tools, functions and functionalities of the platform and how to manage these.

A support centre will provide additional help. The "Help"-Menu provides a telephone number that can be called at a certain period of time. A person at the Call centre will answer the questions of the users. He/she will also be able to remotely access the user's

computer over the network– if the user has activated this option in the user setting- to resolve any technical problems.

4.2 Security and authentication issues

The Join-In social network is designed as a private network, which allows access for registered users only. To achieve acceptance of the users obeying the users right on privacy and data protection is vital. This policy is addressed in the *Join-In Social Media Privacy Policy* which is being developed within the Join-In project.

4.2.1 Authentication

The user has different options to get authenticated to the Social Network:

- Paper form: A user can register in the Join-In Platform filling a form in one of the regional user centres. Here the User Support will give the login data to the user and help the user with the registration.
- Email: For users who own an email account. The user can access to a registration form within the Join-In Portal, where he will be asked for “User Name”, “Email” and “Password”. Once the User Admin has confirmed the registration process, the user will get a confirmation email, allowing him/her to access the portal.

The user need to login on to the portal using “User Name” and “Password”. To avoid inserting the login every time, the information can be stored in the user’s machine.

This option allows the user to use different devices to access the web-based portal.

- Pen Drive Security / Dongle: The credentials are stored in this physical device. The user connects it to the client machine and the Dongle launches a client browser and proceeds to add the access information.
- MAC: The registration can take place when the user buys the device (STBs, All-In-One, etc…). The MAC address is registered additionally to the user information. This address can work as user identification, avoiding the need of inserting the login information every time, but not allowing the client to access the platform from different computers.

  This last option is not secure, because hostile users can mask the MAC, replacing the legit owner of the account. It does, therefore, not provide an option for Join-In.

5 User environment devices

This chapter describes the hardware devices to be used by the users at home in order to take part in the Join-In social network and on-line games and services.

Join-In offers various user device options depending on the requirements of the end-users. Some users may use a simple-to-use set-top box and a robust controller, while others more experienced user or peers can access the Join-In social network using a multi-touch All-in-One PC, tablet or a PC.

5.1 Set-top boxes

Two set-top box approaches were defined. One focuses on embedded solutions with DVB reception capabilities and Web access, so called hybrid set-top boxes. The second approach, the streaming set-top box, is with mere Web access and optimized streaming capabilities. TV content is accessible via portals like zattoo.com.

No special measures are taken into account for digital rights (DRM) or conditional access (CA) management of general media content. The user can access free to air (FTA) and free Web content. Upgrades to the reception of protected media are available by end-users' requests.

Set-top box specification:

Common Features

- WLAN and wired network connection
- Bluetooth
- HDMI, SCART and video/audio connectors
- Supported resolutions Full HD 1920x1080 and PAL 720x576 (DV) 768x576 (DVB)
- Internal or external web socket server for motion data access from the embedded browser
- External Webcam
- Virtual keyboard (wireless keyboard optionally);
- Wireless mouse or Join-In controller

Streaming Set-top box

- Pentium dual core or AMD E450 with graphics for DirextX support,
- Windows 7 performance index ≥ 3.9
- Operating System
- Windows 7 with DirectX 11 or Ubuntu 12.04
- Browser with HbbTV add-on

Hybrid Set-top box (optionally hybrid TV)

- DVB-tuner (S, C or T)
- Linux kernel version 3.2
- Embedded HTLM5 Browser based on Webkit, Opera or others
- Standard HID mouse driver
- HbbTV ≥ Version 1.5
- OpenGL ES 2.0 (WebGL) support (expected in next gnereration)

5.2 All-in-one multi-touch PC

Join-In is also accessible with all-in-one touch-screen PC as end-user devices. These are all-in-one devices integrating the processing unit in the screen itself. It is fully possible to dedicate the usage of such a PC for special purpose applications (like Join-In) and having the user control the functionality using only the touch-screen or another dedicated controller. The touch-screen PCs are supported by Linux and Windows, and have multiple input/output possibilities making it feasible to connect to other devices.

The minimum requirements for Join-In will be

- All-in-One multi-touch Pentium dual core or AMD E450 with graphics for DirectX and WebGL support
- Windows 7 performance index ≥ 4
- Windows 7 with DirectX 11, or Ubuntu 12.04

5.3 Tablets & Smartphones

Join-In will to some extend support Tablet and Smartphone as end-user devices. However, for some Join-In services the specification of a Tablet or Smartphone may not meet the minimum requirements by that game or service. There may also be issue regarding processor speed, graphical acceleration, operating system, web browser characteristics, and screen size.

Though, some games and services, like the biking exergame of Join-In, will be designed and implemented specially for Tablet devices. The candidate tablets are the Android tablet or Apple iPad.

5.4 PC / Laptops

Join-In will also be accessible and support with standard PC's and laptop computers as end-user devices. However, it is not envisioned that this will be the primary user-device of choice for the Join-in target group.

5.5 Microsoft Kinect

In Join-In we will use the Microsoft Kinect for detecting the user movements in several of the games and exergames that are being developed.

Kinect for Xbox 360 and Kinect PC is a game controller launched by Microsoft. Using advanced artificial vision technologies it is capable to recognize 3D movements with the highest precision by just using its camera.

Microsoft has not made any attempt to limit the use of the technology for other purposes, and the open source community has already developed open source APIs for the device.

In Join-In we will access the Kinect data and functionality by using either;

- Valentia Kinection open source community project. The service currently employs Microsoft Kinect SDK and is thus restricted to end-user devices running MS Windows.
- SANDRA cross-platform application library for controllers and motion sensing input devices.

5.6 Exerbike sensors

The exerbiking game will be designed and implemented so that it both works with and without external activity sensors. Candidate external sensors can be: simple step counters, the exerbike built-in speedometer, a dedicated bike-attached sensor, or even the users Smartphone strapped to their leg. The choice of activity sensor will depend on the need for the actual biking game of getting detailed up-to-data data on the performance of the biker.

The default setting for exerbiking will be to use the tablets internal activity sensor to detect the vibration of the exerbike as a measure of biking activity.

5.7 Scoop pointing device

The Scoop is an air-pointing device available from Hillcrest Labs[9] that will be used as reference device for comparison with the Join-In controller.

5.8 The Join-in controller

Navigating through all of the functions and applications offered in a Browsers environment requires more than the traditional remote control buttons. The keyboard and mouse commonly used with PCs are not suited if the user is sitting on a couch or moving in the room. A multi-touch screen offers additional comfort but does not give the freedom of moving around. The CE industry is now offering controllers with different options to overcome the drawbacks of the classical controllers in a hybrid media environment. A lot of research has been done but feedback from use by users in the real world is just starting to emerge.

For example the Kinect and WiiMote perform well in a gaming environment but are not suited for complex menus. Also the approach of using smartphones and tablets as remote controllers based on suited Apps brings added complexity to the home of the elderly, requiring experience with such devices and installation knowledge. Currently such devices are seen as an additional option but will not fully substitute the need for dedicated remote controllers.

Remote control functions offered are based on different methods

- Classical Key input
- Scroll wheel, wobbling knob, touchpad or touch slider for improved cursor control

[9] http://hillcrestlabs.com

- Gyro based motion sensing for cursor control (scoop, Google remote)
- IR camera based motion sensing (leap motion, Kinect)
- Additional keyboard on the back side for text input
- Voice recognition mainly as substitution for text input (ruwido r117, etc)
- Gesture recognition based on gyro or IR camera data processing
- Special forms of gestures are switching through turning the device and shaking gestures

Join-In will therefore in addition to the functionality available with the Kinect also develop a gyro based approach for browser control.

For initial research on gyro-based controller the Scoop Remote Pointer with an USB-wireless adapter was used. Optimal shape, weight, location of tactile switches and sliders was evaluated with the user groups. The result will be a Join-In controller for browser navigation suited for the elderly, where future extensions for voice recognition are taken into account.

The Join-in controller will be an air-pointing device with wireless motion data transfer.

- Proprietary wireless connection with usb-adapter without pairing requirements or Bluetooth connection
- Gyro sensor with at least 6 axis for linear xyz acceleration and orientation (pitch , roll, yaw)
- No recalibration requirements, magnetometer for drift compensation
- 360° of freedom
- Low latency feedback
- Tremor cancellation
- Multiple device operation supported
- HID mouse device function supported
- Low weight
- Optimized power management
- Adaptable control buttons allow adjustment to different skill levels via press buttons, rocker switches and others
- Provided with reference kit and websocket server for access through HTML5 browser applications

Fig. 4 Demo Module

Websocket Server/ Basic Sensor

Fig. 5 Preliminary example of controller design

6 Game software components and services

6.1 Game Servers

The game servers are written in Node.js (http://nodejs.org), which is a server-side software system for developing scalable network applications. Node.js is both a runtime environment and a library. The runtime environment is written in C and provides a context that enables a developer to run JavaScript code outside of a web browser environment. The library features modules that provide APIs for network I/O, file system I/O, process I/O and other services relevant for writing web applications.

The game servers also use a framework for Node.js called Socket.IO (http://socket.io), which enables the creation of real-time web applications. Socket.IO runs on both the server side on Node.js and also on the client side in the web browser. Socket.IO provides an abstraction layer over Web Sockets and other communication schemes, depending on the browser capabilities.

Together, Node.js and Socket.IO provide a scalable solution for building non-blocking HTTP servers with the advantage of using a dynamic scripting language for rapid prototyping.

6.2 Game Software Clients

The game software clients are written in JavaScript and run inside a web browser environment. The Socket.IO library as discussed above handles Client/server communication.

6.3 Motion-sensor and controllers servers

In order to input a users' movement data into the games, we exploit sensors such as the Microsoft Kinect, the Nintendo WiiMote and the Scoop Remote Pointer from Hillcrest Labs. In addition a Join-In controller is being developed.

The data from these sensors has to be gathered locally in the end user device, and then made available to the game client or game server. This is done by running a local server that handles the data produced by the motion sensor devices and controllers, and then make the information available to game clients, games servers or social services using web sockets.

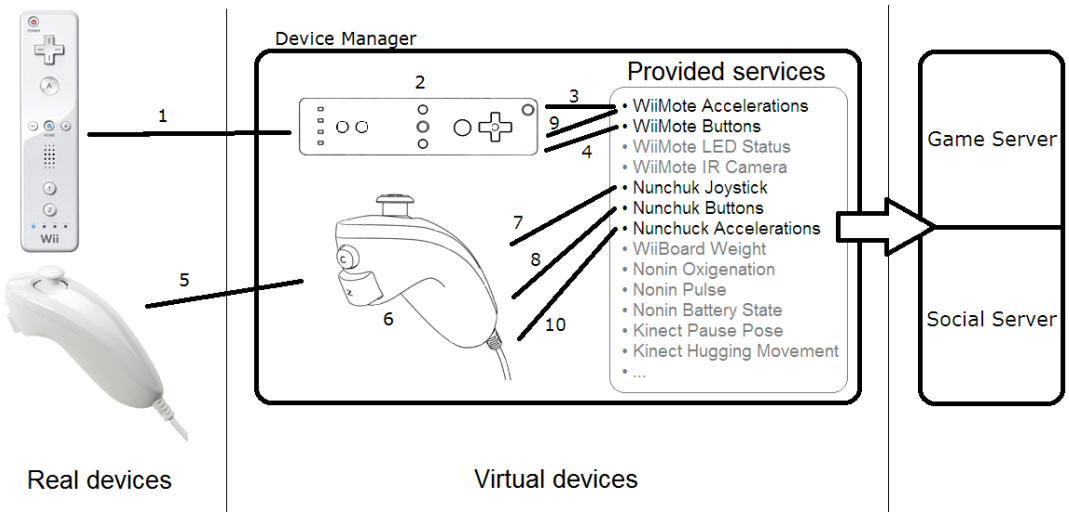
6.4 A cross-platform application library for controllers and motion sensing input devices - SANDRA

We can exploit sensors such as the Microsoft Kinect or the Nintendo WiiMote in order to input a users' movement data into the games. The following section describes the structure and dependencies of the SANDRA application library, which provides access to these sensor devices locally and sends the data to a remote game server or social server.

SANDRA is an easy-to-use facade exploiting a multi-threaded event-based Java library called CommModule. The CommModule contains the logic for;

- Communications with the remote game- or social server.
- Handling of the data produced by the motion sensor devices and controllers
- Upper-level functionalities which goes beyond the pure raw data detection

SANDRA currently supports the following devices: Microsoft Kinect, WiiMote, WiiMotion Plus, Nunchuk, WiiBalanceBoard, and Pulsioximeter Nonin 9650. For instance, the CommModule library can process a user's Kinect skeleton joint data to detect whether that user is standing still, walking or running.



On the picture above we can see a simplified example of how this library works. Let's say we have a WiiMote and its Nunchuk. When the WiiMote is connected (1), the

DeviceManager creates a new virtual object (2). This object will provide acceleration events (3) and button pressed service events (4) both belonging to the WiiMote sensors. Later on, the Nunchuk is connected (5). Then, the DeviceManager creates a new Nunchuk virtual object (6). This object will throw acceleration events (7) and analogic joystick events (8) belonging to the Nunchuk itself. Next, another application requires acceleration data from the WiiMote and, therefore, a new WiiMote acceleration event thrower is linked to the virtual device (9). Finally, another application wants to get data from the Nunchuk analogic joystic, so a new event thrower of that type is added (10).

We use web sockets to perform all the communications between CommModule and the external server. Particularly, we use a thread wich uses the java.net.socket library to handle the input-output operations. It connects to a given IP+Port address and keeps an open channel for data streams. All information, which is sent to the server, is encoded in the JSON format. Therefore, each packet containing the provided service data is a single form-variable JSON string.

SANDRA libraries

There is a set of libraries, which CommModule depends on. We are going to go through them one by one:

KinectLibrary

This library provides the users' skeleton joint data and the Kinect motor and LED control. It uses libusbjava and OpenNI to access, control and retrieve the Kinect data. The libusbjava is a Java wrapper for the libusb 0.1 and libusb-win32 library. It is the responsible for granting easy access to the Kinect through the USB port on any platform. The OpenNI multi-language cross-platform framework which we use is a home-modified version of the latest Java unstable release, making it thread-safe. It is worth saying that the OpenNI framework needs three different modules to be installed before it can be used. The first one is the OpenNI module itself. It has all the interfaces to interact with both the Kinect and the middleware components (e.g. body tracking). The sencond one is the avin2 SensorKinect, which is the driver of the Kinect. The third one is the NITE middleware which contains the algorithmic for the OpenNI functionality in terms of skeleton segmentation, tracking and machine vision. KinectLibrary also uses the j3dcore library and vecmath for the skeleton joint tracking algorithm defintion.

WiiboardSimple

It is a library which enables users to utilise the Nintendo Wii Balance Board providing the weight that is applied on each of the four areas of the board. It uses the bluecove Blotooth protocol to communicate with the device.

IRGlanceLibrary

This library measures the frecuency of apparerances of infrared points which enter in the WiiMote camera scope. It can be used with different WiiMote controller libraries, but it currently works with the wiigee-plugin-wiimote & wiigee-lib libraries. IRGlanceLibrary allows “computerising” common fitness machines which have cyclical movements: just add some IR Leds to the moving pieces and point the WiiMote camera towars that element.

In addition, the wiigee-plugin-wiimote & wiigee-lib libraries are also used for CommModule WiiMote handling functions. This wiigee-lib library is an Java open-source gesture recognition library for accelerometer-based gestures specifically developed for the Nintendo Wii remote controller. The wiigee-plugin-wiimote is a complement for wiigee-lib providing interfaces to handle the acceleration, rotations, and button events of a WiiMote and the Nunchuck. However, the version we use is a tweaked version for our own project and it is not available anywhere else. Combining these two libraries, we can retrieve the following data: the acceleration along the 3 axes for the WiiMote and Nunchuk, the state of their buttons, the gyroscope events of the WiiMotion Plus, the analogic joistic position and the IR point’s position in the IR camera.

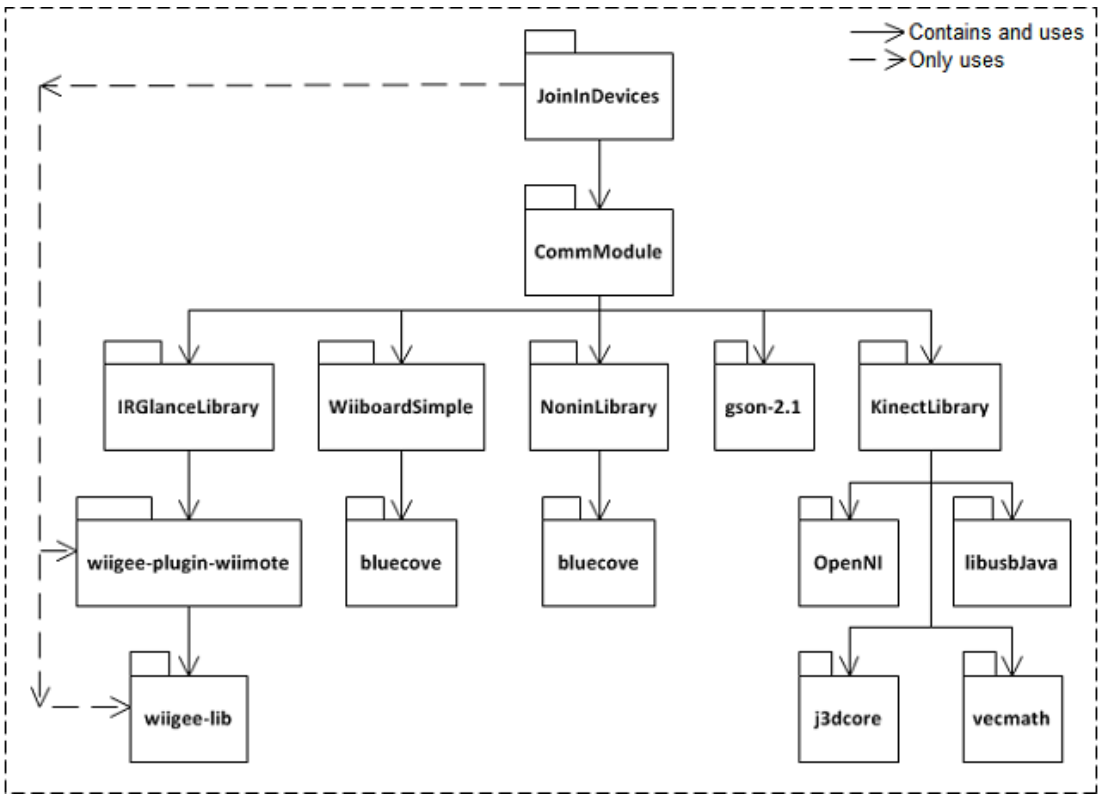
gson-2.1

Gson is a Java library, which can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object. Gson can work with arbitrary Java objects including pre-existing objects that you do not have source-code of.

NoninLibrary

This library uses a BTSPP communication protocol over the Bluetooth stack vluecove and it provides continuous sampling of the Nonin 9650 pulsioximiter. The most useful data provided are: blood oxygenation, pulse, whether the finger is properly introduced in the device or not and the state of the battery.

All the previous libraries dependencies are showed in the following image.



6.5 Valentia Kinection

Valentia Kinection is open source community project. Purpose of Kinection is to provide user ability to publish Kinect data on socket, which then can be utilised on remote clients or HTML5 web pages using socket.

Valentia Kinection serves as a pathway for transmission of data from Kinetic device to the web client. The service extracts data from the device and helps publish it on web sockets, which can then be accessed on web clients via HTML 5 supporting browser such as Safari. Kinect Service Manager operates as a client/server application that provides a platform to the three basic components of the service to communicate with each other, which are:

- Kinect device
- Kinection utility
- Web client

Kinection first utilizes the attached Kinect sensor to accumulate image, depth and skeleton data, then filters this data and publishes it on web sockets. The service employs Microsoft Kinect SDK, which offers drivers and rich APIs for raw sensor streams and human motion tracking, to extract the image, depth and skeletal data from the sensor and publishes this

data on the web using the SuperWebSocket. The web client caters for viewing Kinect data both in the form of Raw Data (joints ID) and Visual Skeleton (skeletal image).

Kinection allows you to stream Kinect color, depth and skeleton from one PC to another PC or a Windows Phone or iPhone/iPad or any other compatible device via websockets. Please review the client sample page for more details on how to use Kinection.

- Project Information URL: http://kinection.codeplex.com/
- Project Download URL: http://kinection.codeplex.com/releases/view/86924
- Project Source URL: http://kinection.codeplex.com/SourceControl/list/changesets

7 Summary

This document has presented the technology components that make up the platform for implementation of the Join-In social network, and described how these components are integrated with each other according to the Join-In social architecture.

The source code of all the components is available on the Join-In source code accounts.

Appendix A- Join-In Connector API

Join-In Connector APIs are REST based, and any client that can send HTTP GET and POST requests can easily consume the API.

How you test the API simply using browser?

Join-In Connector APIs are REST based and it use two HTTP verbs including GET and POST. Using the address bar any browser can be used to test the GET request. Testing the POST request is a bit tricky but there are free open source applications like Curl and Fiddler to test the POST request. For testing the API on windows platform Fiddler is recommended tool.

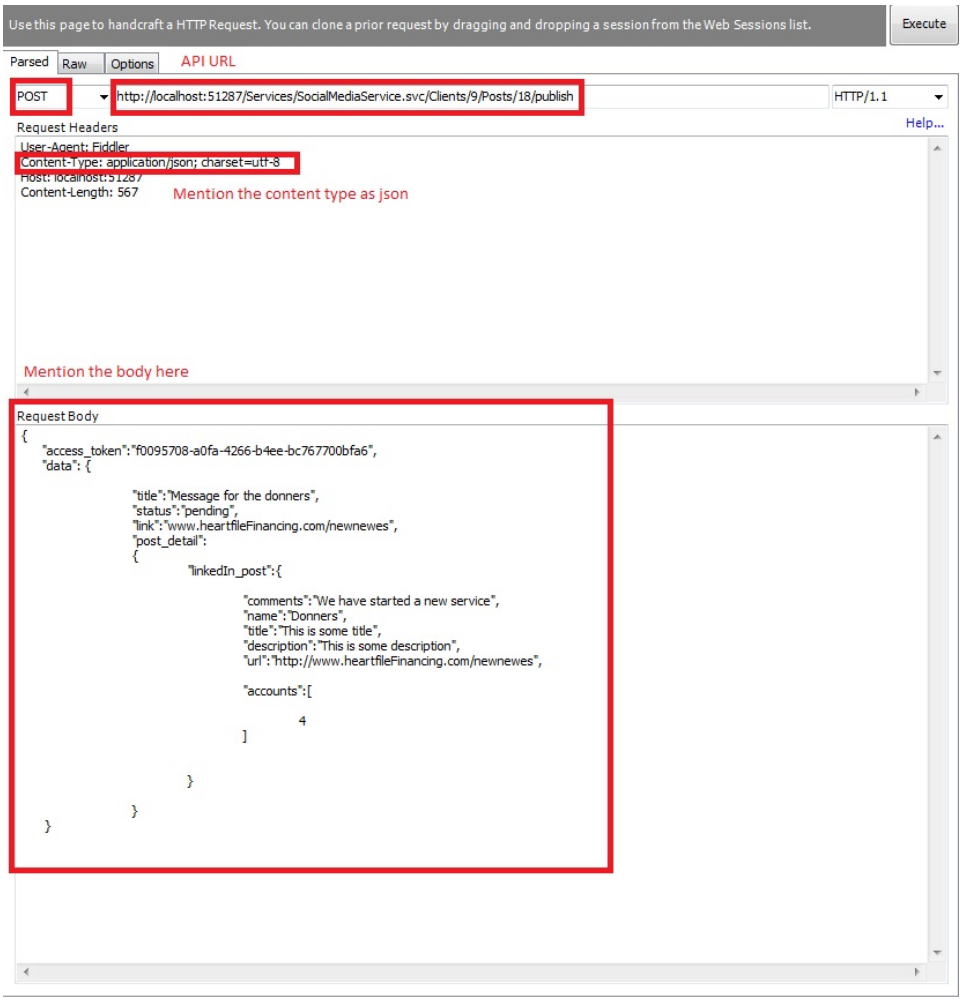


Fig. 6 Fiddler Post Request.

Follow the steps below to send a POST request to Social Dolphin API using Fiddler.

1. Chose “POST” as HTTP verb beside address bar.
2. Enter the URL of the API URL in the address bar.
3. Enter “Content-Type: application/json; charset=utf-8” in the Request Headers textbox as Join-In Connector API returns JSON and Enter “Content-Type: application/xml; charset=utf-8” API returns XML.
4. Enter the request body in the “Request Body” textbox.

Provider Authentication

Every API call needs an access token. To get an access token a user needs to authenticate first.

URL	/provider/GetAuthenticate?username={username}&password={password}
Request Type	GET
Parameters	Required Parameters username: User name of the user password: Password of the user
Request Body	N/A

JSON Result	{
	“Data”:
	{
	“access_token”:””, // A valid access token
	“Id”: , // user id
	“ticks”: , // No of seconds the access token is valid till.
	“username”:”” , // Username of the user
	“displayname”:””
	}
	Message:”” // A message in case any exception/Error
	}

XML Result	<Data> <access_token></access_token> <id></id> <ticks></ticks> <username> </username> <displayname></ displayname> </Data> <Message></Message>
Example	
Request	/provider/GetAuthenticate?username=peter&password=123456
Request Body	N/A
JSON Result	{

	“Data”:
	{
	“access_token”: "ca4e0bdf-1e29-4f54-b015-49351bc8a0ae",
	“id”: 7,
	“ticks”: 6000,
	“username”: "peter",
	“displayname”:”Peter Nilson”
	}
	“Message”: null
	}

XML Result	<Data> <access_token>ca4e0bdf-1e29-4f54-b015-49351bc8a0ae</access_token> <id>7</id> <ticks>6000</ticks> <username>peter</username> <displayname>Peter Nilson</ displayname> </Data> <Message></message>
Comments:	It returns the given information if the user is authenticated, if not error message is returned.

Register a Game Provider

This API allows registering of a new games provider. Each game provider will have their own account and they can use their account detail to login and add new game e.g. Carlow-IT, Valentia.

URL	/Provider/PostAddGameProvider
Request Type	POST
Parameters	Required Parameters username: password: access_token: A valid access token

Request Body	{
	"access_token":"",
	"Data":
	{
	"name": "", // Name of the Provider
	"username": "", // User name for the Provider
	"password": "", // Password of the Provider
	"description": "" // Description of the Provider
	}
	}

JSON Result	{
	"Data":
	{
	"id": // Returns ID of the newly registered provider
	},
	"Message": “” // Message in case of any exceptions/error
	}

XML Result	<Data> <id></id> </Data> <Message></Message>
Example:	
Request	/Provider/PostAddGameProvider
Request Body	{

	"access_token":"ca4e0bdf-1e29-4f54-b015-49351bc8a0ae",
	"Data":
	{
	"name": "Peter Nilson",
	"username": "peter",
	"password": "password",
	"description": "This is some description about the provider"
	}
	}

JSON Result	{
	"Data":
	{
	"id":20
	},
	"Message":null
	}

XML Result	<Data> <id>20</id> </Data> <Message></Message>

Register a Game

This API allows game providers to register a new game e.g. Bubble Break, Walking Game.

URL	/Games/PostAddGame
Request Type	POST
Parameters	Required Parameters name: game name access_token: A valid access token provider_id: provider of the game

Request Body	{
	"access_token":"",
	"Data":
	{ "name": "", // Name of the game
	"redirect_url": "", //url of the game
	"provider_id": "", //id of the game provider
	"Active": "", // Is game active
	"description": "" // Description of the game
	}
	}

JSON Result	{
	"Data":
	{
	"id": // Returns ID of the new game
	},
	"Message":"" // Message in case of any exceptions/error
	}

XML Result	<Data> <id></id> </Data> <Message></Message>
Example:	
Request	/Games/PostAddGame
Request Body	{ "access_token":"ca4e0bdf-1e29-4f54-b015-49351bc8a0ae", "data":

	{
	"name": "My Game",
	"redirect_url": "http://mygamedomain.com",
	"provider_id": 20,
	"active": 1,
	"description": "This is some description about the game"
	}
	}

JSON Result	{
	"Data":
	{
	"id":20
	},
	"Message":null
	}

XML Result	<Data> <id>20</id> </Data> <Message></Message>
Comments:	

List Games

This API allows listing all games.

URL	/Games/GetGameList?access_token={access_token}
Request Type	Get
Parameters	Required Parameters access_token: A valid access token
Request Body	N/A

JSON Result	{
	"Data":
	[{
	"id":, // Id of the game
	"name":"", // name of the game
	"description":"", // description of the game
	"redirect_url":"",
	"provider_id": ,
	"Active":
	}],
	"Message": // Message in case of any exceptions/error
	}

XML Result	<Data> <id></id> <name></name> <description></description> <redirect_url></redirect_url> <provider_id></provider_id> <Active></Active> <Data> . . <Message></Message>
Example:	

Request	/Games/GetGameList?access_token=ca4e0bdf-1e29-4f54-b015-49351bc8a0ae
Request Body	N/A

JSON Result	{
	"Data":
	[{
	"id":20, // Id of the game
	"name":"My game" , // name of the game
	"description":"this is game description", // description of the game
	"redirect_url":"http://gamedomain.com" ,
	"provider_id": 11,
	"Active": 1
	}],
	"Message": // Message in case of any exceptions/error
	}

XML Result	<data> <id>20</id> <name>My Game</name> <description>this is game description</description> <redirect_url>http://gamedomain.com</redirect_url> <provider_id>11</provider_id> <Active>1</Active> </data> <message></message>
Comments	Returns list of games as array of objects

Play Game

URL	/Games/GetPalyGame?gameid={gameid}&access_token={access_token}
Request Type	Get
Parameters	Required Parameters access_token: A valid access token
Request Body	N/A

JSON Result	{
	"Data":
	{
	"text":"" // URL of the game
	},
	"Message":"" // Message in case of any exceptions/error
	}

XML Result	<Data> <text></text> <Data> <Message></Message>
Example:	
Request	/Games/GetPalyGame?gameid=20&access_token=ca4e0bdf-1e29-4f54-b015-49351bc8a0ae
Request Body	N/A

JSON Result	{
	"Data":
	[{
	"text":"http://myurlofgame.com" // URL of the game
	}],
	"Message": // Message in case of any exceptions/error
	}

XML Result	<Data> <text> http://myurlofgame.com </text>

	</Data> <Message></Message>
Comments	Returns URL of the game

Add a Game Player

This API allows registration of a new player.

URL	/Player/PostAddGamePlayer
Request Type	POST
Parameters	Required Parameters username: password: access_token: A valid access token

Request Body	{
	"access_token": "f3f7d6ae-e696-11e1-a3a5-001cc0f96fb6",
	"data": {
	"name": "",
	"email": "",
	"username": "",
	"password": "",
	"photo": "",
	"game_tag": "",
	
	"active":,
	
	"profileoptions": [{
	"profilekey": "",
	"profilevalue": "",
	"profileaccess": ""
	}]
	
	}
	
	}

JSON Result	{
	"Data":
	{
	"id": // Returns ID of the player
	},
	"Message": // Message in case of any exceptions/error
	}

XML Result	<Data> <id></id> </Data> <Message></Message>
Example:	
Request	/Player/PostAddGamePlayer
Request Body	{

	"access_token": "f3f7d6ae-e696-11e1-a3a5-001cc0f96fb6",
	"data": {
	"id": 0,
	"name": "Peter Nilson",
	"email": "peter@my.web.pk",
	"username": "peter123",
	"password": "1234567",
	"photo": "http://peter.com",
	"game_tag": "adventure",
	
	"active":1,
	
	"profileoptions": [{
	"profilekey": "description",
	"profilevalue": "my Description",
	"profileaccess": "-2"
	}]
	
	}
	
	}

JSON Result	{
	"Data":
	{
	"id":20
	},
	"Message":null
	}

XML Result	<Data> <id>20</id> </Data> <Message></Message>
Comments	*Possible profile keys for profile options.* 'description', 'briefdescription', 'location', 'interests', 'skills', 'contactemail', 'phone', 'mobile', 'website', 'twitter'

Player Authentication

Every API call needs an access token. To get an access token a user needs to authenticate first.

URL	/player/GetAuthenticate?username={username}&password={password}
Request Type	GET
Parameters	Required Parameters username: User name of the user password: password of the user
Request Body	N/A

JSON Result	{
	Data:
	{
	“access_token”:””, // A valid access token
	“id”: , // user id
	“ticks”: , // No of seconds the access token is valid till.
	“username”:”” , // Username of the user
	“displayname”:””
	}
	Message:”” // A message in case any exception/Error
	}

XML Result	<Data> <access_token></access_token> <id></id> <ticks></ticks> <username> </username> <displayname></ displayname> </Data> <Message></Message>
Example	
Request	/provider/GetAuthenticate?username=peter&password=123456
Request Body	N/A

JSON Result	{
	data:
	{
	“access_token": "ca4e0bdf-1e29-4f54-b015-49351bc8a0ae",
	“id”: 7,
	“ticks”: 6000,
	“username”: "peter",
	“displayname”:”Peter Nilson”
	}
	Message: null
	}

XML Result	<Data> <access_token>ca4e0bdf-1e29-4f54-b015-49351bc8a0ae</access_token> <id>7</id> <ticks>6000</ticks> <username>peter</username> <displayname>Peter Nilson</ displayname> </Data> <Message></message>
Comments:	It returns the given information if the user is authenticated, if not error message is returned.

Player List

Every API call needs an access token. To get an access token a user needs to authenticate first.

URL	/player/GetPlayerlist?access_token={access_token}
Request Type	GET
Parameters	N/A
Request Body	N/A

JSON Result	{
	Data:
	[{
	“id”: , // user id
	“email”:”” ,
	“name”:””,
	“username”:”” , // Username of the user
	}]
	“Message”:”” // A message in case any exception/Error
	}

XML Result	<Data> <id></id> <email></email> <username> </username> <name></name> </Data> . . <Message></Message>
Example	
Request	/provider/GetAuthenticate?username=peter&password=123456
Request Body	N/A

JSON Result	{
	data:
	[{
	“id”:7,
	“email”: “peter@my.web.pk”,
	username: "peter",
	name:”Peter Nilson”
	}]
	Message: null
	}

XML Result	<Data> <id>7</id> <email>peter@my.web.pk</ticks> <username>peter</username> <name>Peter Nilson</name> </Data> . . <Message></message>
Comments:	It returns the given information if the user is authenticated, if not error message is returned.

Get a Player

Every API call needs an access token. To get an access token a user needs to authenticate first.

URL	/player/GetPlayerInfo?access_token={access_token}&playerid={playerid}
Request Type	GET
Parameters	Playerid:
Request Body	N/A

JSON Result	{
	Data:
	{
	“id”: , // user id
	“email”:”” ,
	“name”:””,
	“username”:”” , // Username of the user
	}
	“Message”:”” // A message in case any exception/Error
	}

XML Result	<Data> <id></id> <email></email> <username> </username> <name></name> </Data> <Message></Message>
Example	
Request	/player/GetPlayerInfo?access_token= ca4e0bdf-1e29-4f54-b015-49351bc8a0ae&playerid=43
Request Body	N/A
JSON Result	{ data: { “id”:7, “email”: “peter@my.web.pk”, username: "peter",

	name:”Peter Nilson” } Message: null }
XML Result	<Data> <id>7</id> <email>peter@my.web.pk</ticks> <username>peter</username> <name>Peter Nilson</name> </Data> <Message></message>
Comments:	It returns the given information if the user is authenticated, if not error message is returned.

Add contact for a Player

This API allows registration of a contact of a player e.g. son, daughter, colleague etc.

URL	/player/PostAddContact
Request Type	POST
Parameters	

Request Body	{ "access_token": "",
	"Data": {
	"contactid": , //0 for new contact
	"ProfileID": , // Id of the payer
	"elggentityid": , //if contact is elgg user
	"relationshipid": , //1---Friend, 2---Family Member,3---Contact
	"groups": "", // comma separated group Ids
	"contactname": "",
	"address":"",
	"phone1":"",
	"phone2":"",
	"skype1":"",
	"skype2":"",
	"mobile1":"",
	"mobile2":"",
	"email1" :" ",
	"email2" :" ",
	"other1":"",
	"other2":""
	}
	}

JSON Result	{ "Data": { "id":"" // Returns ID of the contact }, "Message": // Message in case of any exceptions/error

	}
XML Result	<Data> <id></id> </Data> <Message></Message>
Example:	
Request	/player/PostAddContact

Request Body	{ "access_token": "a4259ae4-f818-11e1-a64c-001cc0f96fb6",
	"data": {
	"contactid": 0,
	"ProfileID": 55,
	"elggentityid": 0,
	"relationshipid": 1,
	"groups": "1,2,3",
	"contactname": "nawaz",
	"address":"my Own Address",
	"phone1":"232323232",
	"phone2":"4444444444",
	"skype1":"nawazhi",
	"skype2":"nawazhi222",
	"mobile1":"0003333",
	"mobile2":"00044444",
	"email1" :"email1@exm.com",
	"email2" :"email1@exm.com",
	"other1":"other11111111",
	"other2":"other22222222"
	}
	}

JSON Result	{ "Data": { "id":20 },

	"Message":null }
XML Result	<Data> <id>20</id> </Data> <Message></Message>

List online/offline status

This API allows listing of player's online/offline status.

URL	/player/GetOnlineStatus?access_token={access_token}&playerid={playerid}&showfriends={showfriends}&status={status}
Request Type	Get
Parameters	Required Parameters access_token: A valid access token Optional Parameter showfriends: whether show only friend status: status of the players to show playerid: id of the player
Request Body	N/A

JSON Result	{
	"Data":
	[{
	"id":"", // userid
	"name":"", // display name of user
	"username": "",
	"onlinstatus": // online status of the user "1" for offline and "false" for online
	}],
	"Message": // Message in case of any exceptions/error
	}

XML Result	<Data> <id></id> <name></name> <username></username> < onlinstatus ></ onlinstatus > </Data> . . <Message></Message>
Example:	
Request	/player/GetOnlineStatus?access_token= ca4e0bdf-1e29-4f54-b015-49351bc8a0ae &playerid=20&showfriends=1&status=1
Request Body	N/A
JSON Result	{ "data":

	[{
	"id":20, // userid
	"name":"Peter Nilson", // display name of user
	"username": "Peter",
	"onlinstatus": 1 // online status of the user “1” for offline and “false” for online
	}],
	"Message": // Message in case of any exceptions/error
	}

XML Result	<Data> <id>20</id> <name>peter nilson</name> <username>peter</username> < onlinstatus >true</ onlinstatus > </Data> . . <Message></Message>
Comments	Returns list of users as array of data

Add player's interested games

This API allows adding player game preference for example what games they would like to play e.g. Bubble Breaker game etc.

URL	/player/PostInterestedgame
Request Type	POST
Parameters	Required Parameters playerId: access_token: A valid access token gameid:

Request Body	{
	"access_token":"",
	"Data":
	{ "playerid":,
	"gameid":
	}
	}

JSON Result	{
	"Data":
	[{
	"playerid":,
	"gameid":
	}]
	
	"Message": // Message in case of any exceptions/error
	}

XML Result	<Data> <playerid></playerid> <gameid></gameid> </Data> <Message></Message>
Example:	
Request	/player/ PostInterestedgame
Request Body	{ "access_token":"ca4e0bdf-1e29-4f54-b015-49351bc8a0ae", "Data":

	{
	"playerid": 20,
	"gameid": 22
	}
	}

JSON Result	{
	"Data":
	{
	"playerid": "20",
	"gameid": "22"
	},
	"Message":null
	}

XML Result	<Data> <player_id>20</player_id> <game_id>22</game_id> </Data> <Message></Message>
Comments:	

Add player's Game Metrics Data

This API allows adding player game preference for example what games they would like to play e.g. Bubble Breaker game etc.

URL	/player/PostAddMetrics
Request Type	POST
Parameters	Required Parameters playerId: access_token: A valid access token gameid: gamemetrics:

Request Body	{
	"access_token":"",
	"data":
	{ "playerid":,
	"gameid":,
	"gamemetrics":""
	}
	}

JSON Result	{
	"Data":
	{
	"id"://Id of the player whose metrics has been just added
	}
	
	"Message": // Message in case of any exceptions/error
	}

XML Result	<Data> <id></id> </Data> <Message></Message>
Example:	
Request	/player/ PostAddMetrics
Request Body	{ "access_token":"ca4e0bdf-1e29-4f54-b015-49351bc8a0ae", "Data":

	{
	"playerid": 20,
	"gameid": 22,
	“gamemetrics”:”metrics data”
	
	}
	}

JSON Result	{
	"Data":
	{
	"id":20
	},
	"Message":null
	}

XML Result	<Data> <id>20</id> </Data> <Message></Message>
Comments:	

Register ELGG user as Player

This API allows registering elgg users which are not registered as game players.

URL	/player/PostRegisterAsPlayer
Request Type	POST
Parameters	Required Parameters playerId: access_token: A valid access token

Request Body	{
	"access_token":"",
	"data":
	{ "id":
	
	}

JSON Result	{
	"Data":
	{
	"id"://Id of the player whose metrics has been just added
	}
	
	"Message": // Message in case of any exceptions/error
	}

XML Result	<Data> <id></id> </Data> <Message></Message>
Example:	
Request	/player/PostRegisterAsPlayer

Request Body	{
	"access_token":"ca4e0bdf-1e29-4f54-b015-49351bc8a0ae",
	"Data":
	{
	"id": 20
	
	}

	}

JSON Result	{
	"Data":
	{
	"id":20
	},
	"Message":null
	}

XML Result	<Data> <id>20</id> </Data> <Message></Message>
Comments:	

List a Player's Profile Images URL

Every API call needs an access token. To get an access token a user needs to authenticate first.

URL	/player/GetProfileImages?access_token={access_token}&playerid={playerid}
Request Type	GET
Parameters	Playerid:
Request Body	N/A

JSON Result	{
	Data:
	[{
	"imagetype": , // user id
	"image":""
	}]
	}
	"Message":"" // A message in case any exception/Error
	}

XML Result	<Data> <imagetype></imagetype> <image></image> . . </Data> <Message></Message>
Example	
Request	/player/ GetProfileImages?access_token= ca4e0bdf-1e29-4f54-b015-49351bc8a0ae&playerid=43
Request Body	N/A

JSON Result	{
	data:
	[{
	"imagetype":"small",

	“image”: “http://mydomain.com/image.jpg” }] Message: null }
XML Result	<Data> <imagetype>small</imagetype > <image> http://mydomain.com/image.jpg</image> . . </Data> <Message></message>
Comments:	It returns the given information if the user is authenticated, if not error message is returned.

List a Player's Profile Images

Every API call needs an access token. To get an access token a user needs to authenticate first.

URL	/player/GetProfileImagesBinary?access_token={access_token}&playerid={playerid}
Request Type	GET
Parameters	Playerid:
Request Body	N/A

JSON Result	{
	Data:
	[{
	"imagetype": , // user id
	"image":"" //base64 string of image
	}]
	}
	"Message":"" // A message in case any exception/Error
	}

XML Result	<Data> <imagetype></imagetype> <image></image> . . </Data> <Message></Message>
Example	
Request	/player/GetProfileImagesBinary?access_token=ca4e0bdf-1e29-4f54-b015-49351bc8a0ae&playerid=43
Request Body	N/A

JSON Result	{
	data:
	[{
	"imagetype":"small",

	"image": "R0lGODlhGQAZAOYAAGxsbHV1dc3Nzc7OztDQ0NLS0nl5eX5+fnp6enx8fHJycs/Pz3R0dIKCgnd3d29vb3Z2doODg3FxcdTU1Hh4eHt7e9PT02VlZWNjY7e3t2BgYGZmZmhoaG1tbW5ubsvLy4uLi8fHx3BwcGRkZKSkpKqqqq2trZiYmIGBgZ2dnXNzc8DAwJqamrCwsMrKyl9fX729vYSEhImJicjIyMLCwqWlpb+/v4aGhoeHh5eXl11dXaysrNbW1tra2rS0tMXFxa+vr4+Pj8TExIyMjKenp7Ozs5CQkLm5uaCgoH9/f5GRkaurq5ycnJmZmdHR0YCAgGdnZ2lpaWJiYmpqasbGxmtra8zMzAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACH5BAAAAAAALAAAAAAZABkAAAf/gFaCg4SFhoeGAk5OBIyIj1ZOLkUkKTVHBAOQhU4wQ0+goCxWmpukITdJEaurCEgFppElFQ21tk8oNAubAgQnFaHBFS1OvAtKBgfKywcUS7CQiicQCdXWFQYZxZsFQBAI4OEUMR+lkAUtDBQG7OwBMrumTj4qEA739woyBLEENgEMAggMAOGBEWibBsxwlIKBw4BVWEyIFYmIAgkKMj4wsIKfKQELegSZIqFkFRA8Fgh4NKCABQIfMgQA8KAmAAgrFFkosJJQgRkmmOCYWcVDh6MeAIigACKFiV6DnGRQcOHChVAGjdCmAKhw0jMKAoR8qFCgxT0qpdyzbtixwTRi2U0DAlit27ePPa3eAhxIIJMTRwgEK4sOHDhDm8IDEBBgYNUiJLnkx5so4AA5qMqMK5s+fPn6dc2PGDiunTqFOrpiKESiAAOw==”
	}]
	Message: null
	}