**AdVisuo**

**server side**

# by Jarek Francik, jarek@kingston.ac.uk

# Database

The database is already stored on the SQLServer at intranet.rlr-uk.com. It is organised in 8 tables, shown in the diagram in the next page and summarised below:

* **AVProjects**: the main table containing the information about the associated building, the lift journeys and passengers. The relationship between the project and the building is basically one-to-one, however the database structure allows to associate various projects to the same building structure, which may be utilised in the future. Project may be associated with many journeys and passengers. The ready flag indicates if the entire dataset is ready to download (ie. the collections of journeys and passengers are complete), and may be used in the future for "streamed” download of partial datasets.

READ FROM BUILDING INFORMATION XML FILE:

* **AVBuildings**: general information regarding the lobby layout, number of lift shafts etc. In the source XML file those data are labelled as “LobbyLayoutData”. Buildings have many lifts and floors.
* **AVLifts**: dimensions and other parameters of the lifts and lift shafts
* **AVFloors**: dimensions (the height) of the floors and other parameters

READ FROM SIMULATION BINARY FILE

* **AVJourneys**: each record corresponds to a single non-stop journey of a lift car from one floor to another. Each journey has many door cycles.
* **AVJourneyDoorCycle**: each record describes the event of lift doors opening and then closing. Typically, there is one such cycle per each journeys, however sometimes there may be no door cycle (doors do not open) or more than one door cycle (doors re-open). Also, in case of double-decker cars, there are typically double door cycles per journey.
* **AVPassengers**: each record corresponds to a single passenger passing through the system. On the simulation side (AdSimulo), this kind of event is known as “hall call”, which doesn’t seem adequate for the visualisation engine. Each passenger has many way points.
* **AVPassengerWayPoints**: waypoints illustrate consecutive stages in the life of the passenger within the system, like walking, waiting, traversing lift doors etc, described with spatial coordinates and time – used directly to generate the visualisation.



# AdV.dll

This is the main server-side module for the visualisation engine. Functions exposed in this library operate on the building structure XML file and simulation binary files. The results are stored in the database, ready to be sent to the client side. The adv.dll is not directly called by the client side. **It should be called from within the server software**, probably **just after the simulation is completed**.

Main functionality:

* read input files (building structure XML and simulation binary data)
* pre-process the information (identify lift journeys, plan passengers’ behaviour, resolve spatial collisions etc.)
* store the output into the database

Available functions:

* HRESULT AVLoad(AVSTRING pConnStr, AVULONG nProjectId, AVSTRING pBuildingXml, AVSTRING pBinarySim, AVFLOAT fScale = 0.04f);  
  Loads Building Information and Simulation Data, then stores them in database.
* HRESULT AVLoadBuilding(AVSTRING pConnStr, AVULONG nProjectId, AVSTRING pBuildingXml);  
  Loads Building Information and stores the data in database. This function does not load actual simulation data; call AVLoadSim to complete the task.
* HRESULT AVLoadSim(AVSTRING pConnStr, AVULONG nProjectId, AVSTRING pBinarySim, AVFLOAT fScale = 0.04f);  
  Loads Simulation Data and stores then in database. The Building Information must be loaded with AVLoadBuilding function.
* HRESULT AVCleanUp(AVSTRING pConnStr, AVULONG nProjectId);  
  Removes all the information regarding the project identified with *nProjectId* parameter from the database.
* HRESULT AVCleanUpAll(AVSTRING pConnStr);  
  Removes all the information from the database (leaves the database empty).

Parameters:

* pConnStr: connection string.
* nProjectId: project id, to be used later to recall data.
* pBuildingXml: building information XML pathname (this is the file generated by Peter).
* pBinarySim: binary simulation data pathname (this is the file generated by Krzysztof).
* fScale: a value used to scale the building information to the dimensions used within the 3D scene; should normally be left at the default value of 0.04f.

Return Value:

* HRESULT value will typically be either S\_OK if successfully completed, or a standard error value otherwise. See the next section for a method to identify errors. Additionally, AVLoadSim returns S\_FALSE in case the building information is not stored in the database.

Remarks:

* Use mainly AVLoad function that performs the most complete range of actions.
* Project ID’s are not auto-numbers so unique values must be supplied by the caller.
* Do not override a project with another one with the same id: this will break the constrain of primary key uniqueness. If necessary, call *AVCleanUp* first to remove the old data.

# Adv.dll Coding Example

The adv.dll library is rather simple in use. For a more elaborate sample please check the advtest project source code. The code below shows how to call a function and identify possible errors.

CoInitialize(NULL); // obligatory!

try

{

HRESULT h;

h = AVLoadSim(pConn, nProjectID, pSim);

if (FAILED(h)) throw \_com\_error(h);

if (h == S\_FALSE)

wcerr << L"Necessary data missing in the DB. "

<< L"Provide Building XML file as input." << endl;

}

catch(\_com\_error &ce)

{

wcerr << L"Exception thrown for database operation:\n" << endl;

wcerr << L"Code = " << ce.Error() << endl;

wcerr << L"Message = " << ce.ErrorMessage() << endl;

wcerr << L"Source = " << (LPCTSTR) ce.Source() << endl;

wcerr << L"Description = " << (LPCTSTR) ce.Description() << endl;

}

CoUninitialize();

# AdB.dll

This module should be used in order to generate the IFC files.

IFC is an open standard for BIM, Building Information Modelling, supported among others by Autodesk Inc.

Available functions:

* HRESULT AVDB2IFC(AVSTRING pConnStr, AVULONG nBuildingId, AVSTRING pIfcPath);  
  Loads Building Information from the database and converts to the IFC standard.
* HRESULT AVXML2IFC(AVSTRING pXmlPath, AVSTRING pIfcPath);  
  Loads Building Information from an XML file and converts to the IFC standard. No database is used.

Parameters:

* pConnStr: connection string.
* nBuildingId: building ID, typically the same as the corresponding project ID.
* pXmlPath: building information XML pathname (this is the file generated by Peter).
* pIfcPath: path to the output IFC file.

Return Value:

* as in adv.dll

Remarks:

* A 3rd party DLL *IFCEngine.dll* is used.
* A 3rd party file, *IFC2X3\_TC1.exp*, must be made available at the current path.

# AdvTest.exe and AdbTest.exe

These programs are not intended to act as part of the target platform. They are provided as temporary testing tools that may also be used for simple maintenance.

These programs are available as executables and as the source code.

Run the program from the command line, without arguments, to see the usage.

# AdvSrv Web Service

This is the only part of the system that is directly called by the client application, and also the only component written in C#. This Web Service exposes just a single function of the form:

[WebMethod] public DataSet[] GetAVProject(int id);

For the given project id this function returns an array of 8 DataSets, each of them corresponding to one of the database tables. The client side consumes the XML structure sent over HTTP.

This Web Service is very simple (50 lines of source code).

# AdVisuo Project URL

The end user should be able to run his or her client software by clicking on a specially formatted URL. The client software will then automatically connect to the AdvSrv Web Service and download the project data.

Here is the proposed format for the URL:

advisuo://web.service.address/ID/auth-key

where:

* web.service.address is the address of the AdvSrc web service
* ID is the project ID, as stored in the database
* auth-key is for security, and will not be used in the first stage of development.

Example for Project ID = 42:

[advisuo://intranet.rlr-uk.com:8080/advsrv.asmx/GetAVProject/42/74324728749824265](http://intranet.rlr-uk.com:8080/advsrv.asmx/GetAVProject/42/74324728749824265)

# Summary of Requirements

1. Database structure to be created
2. The adv.dll module to be activated after the simulation is complete, prior to the visualisation
3. The adb.dll module to be activated when IFC file is required by the end user
4. **AdvSrv Web Service to be installed in the server and made available for the client software**
5. Download site to be created, from which the end user could download the client software
6. URL advisuo links to be exposed in the web site.

(1) is done. (2) and (3) may be emulated manually by using advtest and adbtest tools. The web service for (4) is up and running in the c:/advisuo/source/advsrv directory, but it still needs to be installed on a public server. The setup package for (5) will be delivered within a week. (6) is entirely the responsibility of the web team.