3D Inspection System

Office Management API

Philosophy

Version 12 of the 3D Inspection System was designed to connect either as a direct SQL connection, or as a cloud connection.

In order to keep all programming consistent, we use a set of services that work with both types of connections.

All communication with the database is asynchronous, since the cloud is using REST with json structures.

Reference

Modules

All libraries are built using Visual Studio .NET 4.0. The following libraries make up the API:

Cloud3D.Models.dll

These are all the data structures that are used for communication.

Cloud3D.UniversalControllers.dll

Universal controllers allow a single API to connect to both a cloud database AND a direct SQL connection.

Cloud3D.SQLControllers.dll

These controllers provide direct access to a SQL database. (always use the universal controllers to guarantee the software works with all databases).

Cloud3D.SQLControllers.Restclient.dll

These controllers provide access to a cloud database. (always use the universal controllers…..)

Supp3DUI.dll

Allows direct access to the 3D Office Management user interface.

Models

Simple models are used to represent all objects in the database.

AuthenticationModel

This model is populated and used to connect to either a SQL or Cloud database.

CompanyModel

Represents a company, such as an individual real estate office. This company may them be attached to contacts (such as a real estate agent.) or an inspector.

ConnectionModel

This model is populated with credentials and used to connect to either a SQL or Cloud database. The model is then used with the Cloud3DConnection.Login method to login to a database.

ConnectionType – either SQL or Cloud

Cloud_UserID, Cloud_Password – credentials for login to cloud.

SQL_ConnectionString – connection string for login to SQL database.

SQL_TenantID – A single multi-tenant database per SQL server is used to store office management data. This ID represents the specific tenant in the database to use.

To make login easier, there are helper methods in Supp3DUI. **TODO**

ContactModel

Represents a single contact in the database.

ContactRoleModel

A role is used to identify the TYPE of contact. Multiple roles may be attached to a contact. For instance, a contact could be a client role AND an agent role.

InspectorModel

Represents a single inspector

ItemModel

Represents an invoice item, such as a service (inspection, radon inspection….).

JobContactModel

Represents a contact attached to a job. For instance, a client and an agent could be attached to a job.

JobInvoiceModel

Represents a single invoice item in the jobs invoice collection.

JobModel

Represents a single job in the database. A job contains all the contacts, contact information, information about the inspection, and could potentially contain either a

single schedule or multiple scheduled inspections (if Multischedule is true in the preferences).

ListItemModel

There are 25 customizable text fields, each of which can have a collection of selectable pull-down items. The ListViewModel represents a single item in the customizable pull down list. You can determine what these customizable fields are using the preferences model.

ListParameters

Is used to represent the criteria for querying a list of objects using the “List” method of a universal controller.

Each “List” method supports different properties of the “ListParameters” model. See the documentation for the universal controllers below for details.

Below are all the properties of ListParameters:

Skip – Some lists support returning a range of results. The “Skip” property represents the first record to be returned in a query of records.

Take – This represents the maximum number of records to return. It can be used along with “Skip” to return a block of records.

FilterCriteria – This is an enumeration for the type of criteria. If the list supports “Name”, the resulting list will consist of all values where the name matches the criteria. If the list supports “General”, the result will contain a list of matching results where the criteria could match multiple fields. For instance, searching for a contact using “General” and “Ab” as the “FilterValue” will result in a list of all fields where either the name, address, company, or city contains the letters “Ab”.

FilterValue – this is the criteria value.

FilterValueTo – if supported, this would be the bottom boundary of the search results.

CustomParam – this would be a custom parameter that could be supported by a list.

LocalPreferencesModel

Contains all the local machine preferences, such as login information and general software settings.

PreferencesModel

This model is populated with all the company preferencces, such as company address, email settings, and custom field settings.

Universal Controllers

Universal controllers are used to perform add, modify, delete, load, and list operations against the database.

When returning a list of records, we use a “ListParameters” model, which contains the criteria for the list. For instance, if we set the

CompanyController

Uses a CompanyModel object to add or modify a record. An ID can be used to load or delete a record. A list method can be used to return a list of companies.

Supported “ListParameters” values for list method:

Skip, Take, Criteria Name, FilterValue

ContactController

Uses a ContactModel object to add or modify a record.

Supported “ListParameter” values for list method:

Skip, Take, Criteria Name, Criteria General, ActiveState, FilterValue, CustomParam (used to return a list for a contact containing a specific role ID).

ContactRoleController

Uses a ContactRoleModel object to add or modify a record.

Supported “ListParameter” values for list method:

Criteria Name, FilterValue

InspectorController

Uses an InspectorModel object to add or modify a record

Supported “ListParameter” values for list method:

Critera name, FilterValue

ItemController

Uses an ItemModel object to add or modify a record

Supported “ListParameter” values for list method:

Critera name, FilterValue

JobController

Uses a JobModel obbjet to add or modify a record

This object does not support a “ListParameter” object, but instead supports a “ReportModel” object, which allows full report-like querying of the job database.

ListItemController

Uses a ListItemModel object to add or modify a record

Supported “ListParameter” values for list method:

Critera name, FilterValue

PreferencesController

Can be used to return or modify the company preferences.

Code Samples

Connecting

The following code can be used to easily establish a connection using the API. A progress screen will be displayed.

Using Supp3dui.dll hides some of the complexity when logging in, as well as loading some useful static objects, such as supp3dui.staticdata.Preferences, which contains all the setup preferences.

```
            // Load local preferences
            Supp3DUI.StaticData.LocalPreferences = Supp3DUI.BusinessObjects.LocalPreferences.Load("");

            // Create an instance of a connection model and populate it with already
established connection settings.
            var cm = new ConnectionModel();
            cm.ConnectionType = Supp3DUI.StaticData.LocalPreferences.ConnectionType;

            if(cm.ConnectionType == ConnectionModel.ConnectionTypeEnum.ConnectionTypeCloud)
            {
                cm.Cloud_URL = Supp3DUI.StaticData.LocalPreferences.ConnectionCloudURL;
                cm.Cloud_UserID = Supp3DUI.StaticData.LocalPreferences.ConnectionUser;
                cm.Cloud_Password = Supp3DUI.StaticData.LocalPreferences.ConnectionPassword;
            }
            else
            {
                var pref = Supp3DUI.StaticData.LocalPreferences;
                cm.SQL_TenantID = Supp3DUI.StaticData.LocalPreferences.ConnectionSQLTenantID;
                cm.SQL_ConnectionString =
Cloud3D.BusinessObjects.Database.ConnectionString(pref.ConnectionSQLServer, pref.ConnectionSQLTrusted,
pref.ConnectionUser, pref.ConnectionPassword, pref.ConnectionSQLProtocol, true);
            }

            // Use the direct 3D login UI to establish a connection
            Supp3DUI.ViewModels.ConnectViewModel vm = new Supp3DUI.ViewModels.ConnectViewModel(cm);
            var dispatcher = new Supp3DUI.MvvmFramework.Windows.WindowsDispatcher(null);
            if (dispatcher.ShowDialog(vm) == true)
            {
                // Login succeeded
            }
```

Adding a Customer

```
            var m = new ContactModel()
            {
                 FirstName = "Joe", LastName = "SomeBody", Active = true
```

```
};


ContactController objController = new ContactController();

objController.AfterAdd += delegate(Object senderAdd, AddEventArgs args)
{
    if (args.ErrorCode == 0)
    {
        ContactModel mRet = (ContactModel)args.Model;
        m.ContactID = mRet.ContactID;
    }
    else
    {
        _errorCode = args.ErrorCode;
        _errorDescription = args.ErrorDescription;
    }

};

objController.Add(m);
```