
# Getting Started with GENESYS™ API

Vitech

**Vitech Corporation**
2270 Kraft Drive, Suite 1600
Blacksburg, Virginia 24060
540.951.3322 FAX: 540.951.8222
Customer Support: support@vitechcorp.com
www.vitechcorp.com



Publication Date: April 2014

## GENESYS API DEVELOPMENT OVERVIEW

The GENESYS architecture was designed with an extensive public Application Programming Interface (API) in mind, providing full programmatic access to repository and project model data. Vitech follows standard, accepted Microsoft .NET framework practices when it comes to managing public interfaces, resulting in a familiar look and feel to external application developers.

Vitech is a strong supporter of API development efforts, both within individual customer organizations and as open-source examples across the user community, in order to extend and fine-tune the benefits offered by the GENESYS platform. Repository and model data are fully available for virtually any application, whether it's reports, dashboards, "connectors" to other data models, or other user-facing solutions that add additional value to an organization's systems engineering efforts.

### .NET Framework Prerequisites

GENESYS is built via Microsoft's .NET 4.0 framework and API applications must utilize the .NET 4.0 or greater runtime to directly reference and use GENESYS API assemblies. For applications using .NET runtimes that are versioned prior to 4.0, service-based approaches can be used.
Sample API projects are available in your GENESYS install directory's Sample folder to assist in understanding key API methods, properties, and techniques. The GENESYS samples provide Microsoft Visual Studio 2010 solution and project files.

### Key GENESYS API Assemblies

Several assemblies are available for reference, offering comprehensive public interfaces to project model data. For full descriptions of public assembly methods, properties, and other functionality, please visit the API reference in the GENESYS online help files.

#### *Vitech.Genesys*

Required for API application development. The Vitech.Genesys assembly provides GENESYS product and version data. While it may or may not be used explicitly, it is required by other API assemblies.

#### *Vitech.Genesys.Client*

Required for API application development. The Vitech.Genesys.Client assembly is the workhorse for working with repository data. It provides access to key data structures such as repositories, projects, folders, entities, attributes, and relationships, as well as to schema meta-data such as entity definitions, relation definitions, and attribute definitions. It also contains repository connection and authentication functionality.

#### *Vitech.Genesys.Common*

Required for API application development. The Vitech.Genesys.Common assembly contains key enumerations and constants that are used throughout the application, such as entity definition identifiers and type descriptors. It also contains definitions used to convert values to GENESYS types, as well as exception classes specific to GENESYS actions. While it may or may not be used explicitly, it is required by other API assemblies.

#### *Vitech.Genesys.Random*

Required for API application development. The Vitech.Genesys.Common assembly hosts algorithms and other functionality used across the GENESYS framework. While it most likely would not be used explicitly, it is required by other API assemblies.

## Licensing Assemblies

The GENESYS licensing assemblies would typically not be used directly for API application development efforts. However, they provide licensing services and validation for the GENESYS framework and are required references for an API application.

Note that while most GENESYS API assemblies are compiled for "Any CPU," some licensing functions require independently compiled 32-bit and 64-bit versions. Including both versions in an application will ensure it works seamlessly in either environment.

1) *Vitech.Genesys.License*

2) *Vitech.Genesys.License.Provider_x86* (32-bit)
   *Vitech.Genesys.License.Provider_x64* (64-bit)

3) *Hasp_net_windows* (32-bit)
   *Hasp_net_windows_x64* (64-bit)

4) *Hasp_windows_82194* (32-bit)
   *Hasp_windows_x64_82194* (64-bit)

   Note: The hasp.windows_82194.dll assembly (32- and/or 64-bit) must be output as a file with your API application. It should not be directly referenced; instead, it should simply be added to a .NET GENSYS API project as an existing file. The .dll's "Build Action" property should be set to "Content" and its "Copy to Output Directory" property should be set to "Copy if newer."

## App.config

An API application's app.config file should contain the following nodes in the appSettings section. The node values provide information to the GENESYS services that are required to authenticate a user's license against the GENESYS version in use.

```
<appSettings>
    <add key="LoggingLogLevel" value="Off"/>
    <add key="ProductCode" value="G03"/>
</appSettings>
```

## Security / Licensing

For API applications, the GENESYS framework handles both security and licensing via a common set of types and methods. A basic approach to accessing a Local repository is shown below; it assumes the key GENESYS assemblies listed above are referenced.
Note that additional repository configurations can be created and accessed by leveraging the RepositoryInfo and ConnectionInfo types. Also, error handling is not included in the sample below simply for brevity's sake.

```
// Create a GENESYS ClientModel instance; license validation will be performed
ClientModel clientModel = new ClientModel();

//  A RepositoryConfiguration instance contains repository connection info,
//  credentials, and connection and login methods; a configuration to a
//  Local repository is added by default when the GetKnowRepositories()
// method is called, and will be available as an item in the returned list
```

```
RepositoryConfiguration                    repositoryConfiguration                    =
    clientModel.GetKnownRepositories()[0];

// Connect to the PC/server as well as the GENESYS services hosting the repository
repositoryConfiguration.Connect();

// Login to the repository configuration, and access the Repository instance
if (repositoryConfiguration.Status == AvailabilityStatus.Available)
{
       GenesysClientCredentials credentials =
          new                GenesysClientCredentials(username,                password,
          AuthenticationType.GENESYS);

       repositoryConfiguration.Login(credentials);
       Repository      repository      =      repositoryConfiguration.GetRepository();

       //                    do                    Repository                    work

       repositoryConfiguration.Logout();
}

clientModel.Dispose();
```

## Key Methods

Once a reference to a connected and available Repository instance is present, several methods can be leveraged to access the bulk of your model data. Key ones are shown in the basic example below.

```
//    Get    the    first    project    in    the    repository's    project    list
IProject project = repository.GetProjects()[0];

// Get the root Functions folder
IFolder folder = project.GetFolder(EntityDefinitionConstants.Function);
string folderName = folder.Name;
string className = folder.GetEntityDefinition().EffectiveName;

//          Get          each          entity          in          the          folder
foreach (IEntity entity in folder.GetEntities())
{
       // An entity definition is also available from an entity itself
       className = entity.GetEntityDefinition().EffectiveName;

       // Get each attribute of the entity
       foreach(IEntityAttributeValue attributeValue in entity.Attributes)
       {
              string attribName = attributeValue.AttributeDefinition.EffectiveName;
             string attribValue = attributeValue.GetValueString();
       }

       // Get each of the entity's relationships and targets
       foreach (IRelationship relationship in entity.GetRelationships())
       {
             IRelationDefinition relationDef = relationship.GetRelationDefinition();
             string relationName = relationDef.EffectiveName;

             foreach(IEntity                    entityTarget                    in
                entity.GetRelationshipTargets(relationDef))
             {
                    string entityTargetName = entityTarget.Name;
             }
       }
```

```
}
```