# APPENDIX C. LIRA API SUMMARY

## General information

If, when working with SP LIRA 10, you need to export initial data or calculation results to a non-standard format or perform additional calculations using existing data, and at the same time you have programming skills, then the information below will be useful to you.

Adding the SP LIRA 10 extension requires two steps:
1. Design an extension based on features.
1.  2. Register the developed extension in the SP LIRA 10.12 environment.

After successful completion of these actions, the **Expansions** menu item will appear in the main menu of SP LIRA 10.12, including sub-items for calling the extensions you implemented (Fig. C.1).

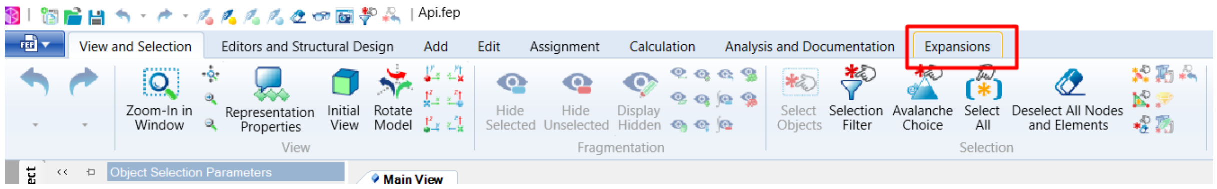

Fig. C.1. Additional menu **Expansions**

✍ *You can implement and register many extensions, for each of them a separate menu sub-item will be generated.*

## Registering the Extension

При первом запуске ЛИРА 10.12 создает файл с глобальными настройками:
When first launched, LIRA 10.12 creates a file with global settings:
```
[ApplicationData]+"\\Lira Soft\\Lira10.12\\VariableEnvironment_x86.xml"
[ApplicationData]+"\\Lira Soft\\Lira10.12\\VariableEnvironment_x64.xml"
```

Among other settings, this file contains the `AddinsPath`, parameter, which contains the path to the folder where the extension registration xml files should be located. By default, this is `[ApplicationData]+"\\Lira Soft\\Lira10.12\\Addins"`.

Extension registration files must have the following structure:
```
<?xml version="1.1" encoding="utf-12"?>
<LiraAddIns>
  <AddIn Type="PROLONGATION">
    <AssemblyPath>path to dll</AssemblyPath>
    <CommandName>command name</CommandName>
    <CommandDescription> description of command </CommandDescription>
    <ImagePath>Path to image of extension icon</ImagePath>
    <Vendor>name of organisation</Vendor>
    <VendorDescription>description</VendorDescription>
  </AddIn>
</LiraAddIns>
```

`AddIn` contains the `Type`, attribute that can take one of three values: `PRIME`, `PROLONGATION` or `ALL`. This attribute indicates in which mode the command of this extension will be available: in the source data editing mode, in the calculation results analysis mode, or in both modes. In the current version, only `PROLONGATION` is available.

`AssemblyPath` is the absolute path to the `*.dll` file.

`ImagePath` absolute path to the file containing the image for the icon in the menu (standard size `Width=24, Height=20`).

`CommandName` s the name of the command in the LIRA 10.12 menu.

`Vendor` and `VendorDescription` – the information about the developer of the extension.

On startup, LIRA 10.12 adds a menu item for each found and successfully read extension registration xml file.

## Extension Development

The recommended development environment for LIRA 10.12 extensions is **Microsoft Visual Studio 2017**. At least two References must be added to the extension project which are the links to libraries from the LIRA 10.12 installation distribution (`LiraAPI.dll` и `FEModel.dll`).

The extension project must implement one `class`, inherited from the interface `LiraAPI.ILiraAPI`, described in the dynamic library `LiraAPI.dll`:

```
public ref class CSampleLiraAPI : public LiraAPI::ILiraAPI
{
            public: virtual LiraAPI::ReturnCodes ExecuteProgram_Result
       (LiraAPI::IResultLiraAPI ^pResultLiraAPI,
            int NodesNumber, int ElementsNumber ,
       List<List<FEModel::Results_Key^>^>^ pAllCases,
      FEModel::Results_Key ^pCurentCase);
}
```

`NodesNumber` and `ElementsNumber` are the number of nodes and elements in the design model.

`pCurentCase` — information about the current load case.

`pAllCases` — information about all load cases available in the task.

`pResultLiraAPI` — is an object that allows you to get tables of calculation results.

The object describing the loading has the following form:

```
ref class FEModel::Results_Key
{
 //index of loading, index of loading history, DCL number,…
 long  m_IndexLoadingCase;
 // index of concurrent loading, DCL variant number,…
 short m_SubIndexLoadingCase;
 // index of form, index of step of nonlinear loading, index of moment of
time,…
 long  m_IndexForm;
};
```

The object `List<List<FEModel::Results_Key^>^>^ pAllCases` contains up to 5 arrays of the form of `List<FEModel::Results_Key^>^`, each of which describes the list of available load cases for different types of tables.

`pAllCases[0]` — load cases and components.

`pAllCases[1]` — eigenvibrations.

`pAllCases[2]` — buckling forms from loadings.

`pAllCases[3]` — design combinations of loads (DCL).

`pAllCases[4]` — buckling forms from DCL.

`LiraAPI::IResultLiraAPI` interface in SP LIRA 10.12 has the form:

```
public interface class IResultLiraAPI
{
virtual int getLiraApiVersion();
virtual DataTable ^get_TableResult(FEModel::e_Results_TableType rtt,
       System::Collections::Generic::List<int> ^pObjArr,
       System::Collections::Generic::List<FEModel::Results_Key ^> ^pKeyArr,
       array<e_Results_ColumnType> ^%pTypeColumns,
       array<System::String ^> ^%pNameColumns);
};
```

`getLiraApiVersion()` function returns the version number of the current LiraApi.

`get_TableResult(…)` function has three input parameters: `FEModel::e_Results_TableType rtt`, `System::Collections::Generic::List<int> ^pObjArr`, `System::Collections::Generic::List<FEModel::Results_Key ^> ^pKeyArr`

and three output parameters: `array<e_Results_ColumnType> ^%pTypeColumns`, `array< System::String ^> ^%pNameColumns` and `DataTable` with query results.

`System::Collections::Generic::List<int> ^pObjArr` is an array of node or element indices (starting from 0)

`System::Collections::Generic::List<FEModel::Results_Key ^> ^pKeyArr` is an array of `FEModel::Results_Key`, objects describing the load cases.

`FEModel::e_Results_TableType rtt` is a type of a table. Detailed information about the available table types can be found in the *LiraAPIHelp.pdf*, ile, which is copied to the hard disk in the `[INSTALLDIR] + "\\LiraAPI"` folder when SP LIRA 10.12 is being installed.

## Extension example

During the installation of SP LIRA 10.12, an archive file with a Visual Studio project is copied to the hard disk in the `[INSTALLDIR] + "\\LiraAPI"` folder, which demonstrates examples with LiraAPI objects.