Компьютерная программа A2v10

Исходный текст (фрагменты)

Автор: Кухтин Александр Анатольевич

Файл DataModelReader.cs

```
// Copyright © 2015-2018 Alex Kukhtin. All rights reserved.

using A2v10.Data.Interfaces;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;
using System.Dynamic;
using System.Reflection;

namespace A2v10.Data
{
    public class DataModelReader
    {

        const String ROOT = "TRoot";
        const String SYSTEM_TYPE = "$System";
        const String ALIASES_TYPE = "$Aliases";

        IDataModel _dataModel;
        IDataLocalizer _localizer;
        IdMapper _idMap = new IdMapper();
        RefMapper _refMap = new RefMapper();
        ExpandoObject _root = new ExpandoObject();
        IDictionary<String, Object> _sys = new ExpandoObject() as IDictionary<String, Object>;

        public DataModelReader(IDataLocalizer localizer)
        {
            _localizer = localizer;
            if (localizer == null)
                throw new ArgumentNullException(nameof(localizer));
        }

        public void SetParameters(SqlParameterCollection prms, Object values)
        {
            if (values == null)
                return;
            if (values is ExpandoObject)
            {
                foreach (var e in values as IDictionary<String, Object>)
                {
                    var val = e.Value;
                    if (val != null)
                        prms.AddWithValue("@" + e.Key, e.Value);
                }
            }
            else
            {
                var props = values.GetType().GetProperties(BindingFlags.Public |
BindingFlags.Instance);
                foreach (var prop in props)
                {
                    var val = prop.GetValue(values);
                    if (val != null)
                        prms.AddWithValue("@" + prop.Name, val);
                }
            }
        }

        Dictionary<String, String> _aliases;

        void ProcessAliasesMetadata(IDataReader rdr)
```

```
{
    _aliases = new Dictionary<String, String>();
    // 1-based
    for (int i = 1; i < rdr.FieldCount; i++)
    {
        _aliases.Add(rdr.GetName(i), null);
    }
}

public void ProcessMetadataAliases(IDataReader rdr)
{
    if (rdr.FieldCount == 0)
        return;
    var objectDef = new FieldInfo(GetAlias(rdr.GetName(0)));
    if (objectDef.TypeName == ALIASES_TYPE)
        ProcessAliasesMetadata(rdr);
}

public IDataModel DataModel
{
    get
    {
        if (_dataModel != null)
            return _dataModel;
        if (_groupMetadata != null)
            _sys.Add("Levels", GroupMetadata.GetLevels(_groupMetadata));
        _dataModel = new DynamicDataModel(_metadata, _root, _sys as ExpandoObject);
        return _dataModel;
    }
}

String GetAlias(String name)
{
    if (_aliases == null)
        return name;
    String outName;
    if (_aliases.TryGetValue(name, out outName))
        return outName;
    return name;
}

void ProcessAliasesRecord(IDataReader rdr)
{
    if (_aliases == null)
        throw new InvalidOperationException();
    // 1-based
    for (int i = 1; i < rdr.FieldCount; i++)
    {
        String name = rdr.GetName(i);
        if (_aliases.ContainsKey(name))
        {
            _aliases[name] = rdr.GetString(i);
        }
    }
}

void ProcessSystemRecord(IDataReader rdr)
{
    // from !
    for (int i = 1; i < rdr.FieldCount; i++)
    {
        var fn = rdr.GetName(i);
        var dataVal = rdr.GetValue(i);
        if (fn == "!!PageSize")
```

```
        {
            Int32 pageSize = (Int32)dataVal;
            _sys.Add("PageSize", pageSize);
        }
        else if (fn == "!!ReadOnly")
        {
            Boolean ro = false;
            if (dataVal is Boolean)
                ro = (Boolean)dataVal;
            else if (dataVal is Int32)
                ro = ((Int32)dataVal) != 0;
            _sys.Add("ReadOnly", ro);
        }
        else
        {
            _sys.Add(fn, dataVal);
        }
    }
}

public void ProcessOneRecord(IDataReader rdr)
{
    var rootFI = new FieldInfo(GetAlias(rdr.GetName(0)));
    if (rootFI.TypeName == SYSTEM_TYPE)
    {
        ProcessSystemRecord(rdr);
        return;
    }
    else if (rootFI.TypeName == ALIASES_TYPE)
    {
        ProcessAliasesRecord(rdr);
        return;
    }
    var currentRecord = new ExpandoObject();
    bool bAdded = false;
    Object id = null;
    Int32 rowCount = 0;
    Boolean bHasRowCount = false;
    List<Boolean> groupKeys = null;
    // from 1!
    for (int i = 1; i < rdr.FieldCount; i++)
    {
        var dataVal = rdr.GetValue(i);
        if (dataVal == DBNull.Value)
            dataVal = null;
        var fn = GetAlias(rdr.GetName(i));
        FieldInfo fi = new FieldInfo(fn);
        if (fi.IsGroupMarker)
        {
            if (groupKeys == null)
                groupKeys = new List<Boolean>();
            Boolean bVal = (dataVal != null) ? (dataVal.ToString() == "1") : false;
            groupKeys.Add(bVal);
            continue;
        }
        AddValueToRecord(currentRecord, fi, dataVal);
        if (fi.IsRowCount)
        {
            if (dataVal is Int32)
                rowCount = (Int32)dataVal;
            else
                throw new DataLoaderException("Invalid field type for !!RowCount");
            bHasRowCount = true;
        }
```

```
            if (fi.IsId)
            {
                if (fi.IsComplexField)
                    _idMap.Add(fi.TypeName, dataVal, currentRecord);
                else
                {
                    _idMap.Add(rootFI.TypeName, dataVal, currentRecord);
                    id = dataVal;
                }
            }
            if (fi.IsParentId)
            {
                if (rootFI.IsArray)
                {
                    AddRecordToArray(fi.TypeName, dataVal, currentRecord);
                    if (!rootFI.IsVisible)
                        bAdded = true;
                }
                else if (rootFI.IsTree)
                {
                    if (dataVal == null)
                        _root.AddToArray(rootFI.PropertyName, currentRecord);
                    else
                        AddRecordToArray(fi.TypeName, dataVal, currentRecord);
                    bAdded = true;

                }
                else if (rootFI.IsObject)
                {
                    // nested object
                    AddRecordToRecord(fi.TypeName, dataVal, currentRecord);
                    if (!rootFI.IsVisible)
                        bAdded = true;
                }
            }
        }
        if (!bAdded)
        {
            if (rootFI.IsGroup)
                AddRecordToGroup(currentRecord, rootFI, groupKeys);
            else
                AddRecordToModel(currentRecord, rootFI, id);
        }
        else
            CheckRecordRef(currentRecord, rootFI, id);
        if (bHasRowCount)
        {
            AddRowCount(rootFI.PropertyName, rowCount);
        }
    }

    public void ProcessOneMetadata(IDataReader rdr)
    {
        if (rdr.FieldCount == 0)
            return;
        // first field = self object
        var objectDef = new FieldInfo(GetAlias(rdr.GetName(0)));
        if (objectDef.TypeName == SYSTEM_TYPE)
            return; // not needed
        else if (objectDef.TypeName == ALIASES_TYPE)
        {
            ProcessAliasesMetadata(rdr);
            return;
        }
```

```
            var rootMetadata = GetOrCreateMetadata(ROOT);
            rootMetadata.AddField(objectDef, DataType.Undefined);
            // other fields = object fields
            var typeMetadata = GetOrCreateMetadata(objectDef.TypeName);
            if (objectDef.IsArray || objectDef.IsTree)
                typeMetadata.IsArrayType = true;
            if (objectDef.IsGroup)
                typeMetadata.IsGroup = true;
            bool hasRowCount = false;
            for (int i = 1; i < rdr.FieldCount; i++)
            {
                var fieldDef = new FieldInfo(GetAlias(rdr.GetName(i)));
                if (fieldDef.IsGroupMarker)
                {

GetOrCreateGroupMetadata(objectDef.TypeName).AddMarkerMetadata(fieldDef.PropertyName);
                    continue;
                }
                if (fieldDef.IsRowCount)
                    hasRowCount = true;
                if (!fieldDef.IsVisible)
                    continue;
                DataType dt = rdr.GetFieldType(i).Name.TypeName2DataType();
                if (fieldDef.IsComplexField)
                {
                    ProcessComplexMetadata(fieldDef, typeMetadata, dt);
                }
                else
                {
                    var fm = typeMetadata.AddField(fieldDef, dt);
                    if (fieldDef.IsRefId || fieldDef.IsArray)
                    {
                        // create metadata for nested object or array
                        var tm = GetOrCreateMetadata(fieldDef.TypeName);
                        if (fieldDef.IsArray)
                            tm.IsArrayType = true;
                    }
                }
            }
            if (hasRowCount)
                _root.AddChecked($"{objectDef.PropertyName}.$RowCount", 0);
        }

        IDictionary<String, GroupMetadata> _groupMetadata;
        IDictionary<String, IDataMetadata> _metadata;

        ElementMetadata GetOrCreateMetadata(String typeName)
        {
            if (_metadata == null)
                _metadata = new Dictionary<String, IDataMetadata>();
            IDataMetadata elemMeta;
            if (_metadata.TryGetValue(typeName, out elemMeta))
                return elemMeta as ElementMetadata;
            var newMeta = new ElementMetadata();
            _metadata.Add(typeName, newMeta);
            return newMeta;
        }

        GroupMetadata GetOrCreateGroupMetadata(String typeName)
        {
            if (_groupMetadata == null)
                _groupMetadata = new Dictionary<String, GroupMetadata>();
            GroupMetadata groupMeta;
            if (_groupMetadata.TryGetValue(typeName, out groupMeta))
```

```
                return groupMeta;
            groupMeta = new GroupMetadata();
            _groupMetadata.Add(typeName, groupMeta);
            return groupMeta;
        }

        void AddValueToRecord(IDictionary<String, Object> record, FieldInfo field, Object
value)
        {
            if (!field.IsVisible)
                return;
            if (field.IsArray)
                record.Add(field.PropertyName, new List<ExpandoObject>());
            else if (field.IsComplexField)
            {
                var propNames = field.PropertyName.Split('.');
                if (propNames.Length != 2)
                    throw new DataLoaderException($"Invalid complex name
{field.PropertyName}");
                var innerObj = record.GetOrCreate(propNames[0]);
                if (value is String)
                    innerObj.Add(propNames[1], _localizer.Localize(value?.ToString()));
                else
                    innerObj.Add(propNames[1], value);
            }
            else if (field.IsRefId)
            {
                var refValue = new ExpandoObject();
                _refMap.Add(field.TypeName, value, refValue);
                record.Add(field.PropertyName, refValue);
            }
            else if (value is String)
                record.Add(field.PropertyName, _localizer.Localize(value?.ToString()));
            else
                record.Add(field.PropertyName, value);
        }

        void AddRecordToGroup(ExpandoObject currentRecord, FieldInfo field, List<Boolean>
groupKeys)
        {
            if (groupKeys == null)
                throw new DataLoaderException($"There is no groups property for
'{field.TypeName}");
            ElementMetadata elemMeta = GetOrCreateMetadata(field.TypeName);
            if (String.IsNullOrEmpty(elemMeta.Items))
                throw new DataLoaderException($"There is no 'Items' property for
'{field.TypeName}");
            GroupMetadata groupMeta = GetOrCreateGroupMetadata(field.TypeName);
            if (groupMeta.IsRoot(groupKeys))
            {
                _root.Add(field.PropertyName, currentRecord);
                groupMeta.CacheElement(groupMeta.RootKey, currentRecord); // current
            }
            else
            {
                // item1 - elemKey, item2 -> parentKey
                var keys = groupMeta.GetKeys(groupKeys, currentRecord);
                var parentRec = groupMeta.GetCachedElement(keys.Item2); // parent
                parentRec.AddToArray(elemMeta.Items, currentRecord);
                if (!groupMeta.IsLeaf(groupKeys))
                    groupMeta.CacheElement(keys.Item1, currentRecord); // current
            }
        }
```

```
        void AddRecordToArray(String propName, Object id, ExpandoObject currentRecord)
        {
            var pxa = propName.Split('.'); // <Type>.PropName
            if (pxa.Length != 2)
                throw new DataLoaderException($"Invalid field name '{propName}' for array.
'TypeName.PropertyName' expected");
            /*0-key, 1-Property*/
            var key = Tuple.Create(pxa[0], id);
            ExpandoObject mapObj = null;
            if (!_idMap.TryGetValue(key, out mapObj))
                throw new DataLoaderException($"Property '{propName}'. Object {pxa[0]}
(Id={id}) not found in map");
            mapObj.AddToArray(pxa[1], currentRecord);
        }

        void AddRecordToRecord(String propName, Object id, ExpandoObject currentRecord)
        {
            var pxa = propName.Split('.'); // <Type>.PropName
            if (pxa.Length != 2)
                throw new DataLoaderException($"Invalid field name '{propName}' for array.
'TypeName.PropertyName' expected");
            /*0-key, 1-Property*/
            var key = Tuple.Create(pxa[0], id);
            ExpandoObject mapObj = null;
            if (!_idMap.TryGetValue(key, out mapObj))
                throw new DataLoaderException($"Property '{propName}'. Object {pxa[0]}
(Id={id}) not found in map");
            mapObj.Set(pxa[1], currentRecord);
        }

        void AddRecordToModel(ExpandoObject currentRecord, FieldInfo field, Object id)
        {
            if (field.IsArray)
            {
                _refMap.MergeObject(field.TypeName, id, currentRecord);
                _root.AddToArray(field.PropertyName, currentRecord);
            }
            else if (field.IsTree)
                _root.AddToArray(field.PropertyName, currentRecord);
            else if (field.IsObject)
                _root.Add(field.PropertyName, currentRecord);
            else if (field.IsMap)
                _refMap.MergeObject(field.TypeName, id, currentRecord);
        }

        void AddRowCount(String propertyName, Int32 rowCount)
        {
            var pn = $"{propertyName}.$RowCount";
            // added in metadata
            // _root.AddChecked(pn, rowCount);
            _root.Set(pn, rowCount);
        }

        void CheckRecordRef(ExpandoObject currentRecord, FieldInfo field, Object id)
        {
            if (field.IsArray || field.IsMap)
                _refMap.MergeObject(field.TypeName, id, currentRecord);
        }

        void ProcessComplexMetadata(FieldInfo fieldInfo, ElementMetadata elem, DataType dt)
        {
            // create metadata for nested type
            var innerElem = GetOrCreateMetadata(fieldInfo.TypeName);
            var fna = fieldInfo.PropertyName.Split('.');
```

```
            if (fna.Length != 2)
                throw new DataLoaderException($"Invalid complex name
{fieldInfo.PropertyName}");
            elem.AddField(new FieldInfo($"{fna[0]}!{fieldInfo.TypeName}"),
DataType.Undefined);
            innerElem.AddField(new FieldInfo(fieldInfo, fna[1]), dt);
        }

        public void PostProcess()
        {

        }
    }
}
```

Файл DynamicType.cs

```
// Copyright © 2015-2018 Alex Kukhtin. All rights reserved.

using System;
using System.Collections.Generic;
using System.Dynamic;
using System.Linq;
using System.Reflection;
using System.Reflection.Emit;
using System.Text;
using System.Threading;

namespace A2v10.Data
{
    public abstract class DynamicClass
    {
        public override string ToString()
        {
            PropertyInfo[] props = this.GetType().GetProperties(BindingFlags.Instance |
BindingFlags.Public);
            StringBuilder sb = new StringBuilder();
            sb.Append("{");
            for (int i = 0; i < props.Length; i++)
            {
                if (i > 0) sb.Append(", ");
                sb.Append(props[i].Name);
                sb.Append("=");
                sb.Append(props[i].GetValue(this, null));
            }
            sb.Append("}");
            return sb.ToString();
        }
    }

    public class DynamicProperty
    {
        string name;
        Type type;

        public DynamicProperty(string name, Type type)
        {
            if (name == null) throw new ArgumentNullException("name");
            if (type == null) throw new ArgumentNullException("type");
            this.name = name;
            this.type = type;
        }

        public string Name
        {
            get { return name; }
        }

        public Type Type
        {
            get { return type; }
        }
    }

    internal class Signature : IEquatable<Signature>
    {
        public DynamicProperty[] properties;
        public int hashCode;
```

```
    public Signature(Object obj)
    {
        Init(GetProperties(obj));
    }

    public Signature(IEnumerable<DynamicProperty> properties)
    {
        Init(properties);
    }

    void Init(IEnumerable<DynamicProperty> properties)
    {
        this.properties = properties.ToArray();
        hashCode = 0;
        foreach (DynamicProperty p in properties)
        {
            hashCode ^= p.Name.GetHashCode() ^ p.Type.GetHashCode();
        }
    }

    List<DynamicProperty> GetProperties(Object obj)
    {
        var props = new List<DynamicProperty>();
        var d = obj as IDictionary<String, Object>;
        foreach (var itm in d)
        {
            if (itm.Value is IList<ExpandoObject>)
                props.Add(new DynamicProperty(itm.Key, typeof(IList<Object>)));
            else if (itm.Value is ExpandoObject)
                props.Add(new DynamicProperty(itm.Key, typeof(Object)));
            else if (itm.Value == null)
                props.Add(new DynamicProperty(itm.Key, typeof(Object)));
            else
                props.Add(new DynamicProperty(itm.Key, itm.Value.GetType()));
        }
        return props;
    }

    public override int GetHashCode()
    {
        return hashCode;
    }

    public override bool Equals(object obj)
    {
        return obj is Signature ? Equals((Signature)obj) : false;
    }

    public bool Equals(Signature other)
    {
        if (properties.Length != other.properties.Length) return false;
        for (int i = 0; i < properties.Length; i++)
        {
            if (properties[i].Name != other.properties[i].Name ||
                properties[i].Type != other.properties[i].Type) return false;
        }
        return true;
    }
}
public class ClassFactory
{
    public static readonly ClassFactory Instance = new ClassFactory();

    ReaderWriterLock rwLock;
```

```
        Dictionary<Signature, Type> classes;
        int classCount;
        ModuleBuilder module;
        static ClassFactory() { }  // Trigger lazy initialization of static fields

        private ClassFactory()
        {
            AssemblyName name = new AssemblyName("DynamicClasses");
            AssemblyBuilder assembly = AppDomain.CurrentDomain.DefineDynamicAssembly(name,
AssemblyBuilderAccess.Run);
            module = assembly.DefineDynamicModule("Module");
            classes = new Dictionary<Signature, Type>();
            rwLock = new ReaderWriterLock();
        }

        public Type GetDynamicClass(IEnumerable<DynamicProperty> properties)
        {
            rwLock.AcquireReaderLock(Timeout.Infinite);
            try
            {
                Signature signature = new Signature(properties);
                Type type;
                if (!classes.TryGetValue(signature, out type))
                {
                    type = CreateDynamicClass(signature.properties);
                    classes.Add(signature, type);
                }
                return type;
            }
            finally
            {
                rwLock.ReleaseReaderLock();
            }
        }

        Type CreateDynamicClass(DynamicProperty[] properties)
        {
            LockCookie cookie = rwLock.UpgradeToWriterLock(Timeout.Infinite);
            try
            {
                string typeName = "DynamicClass" + (classCount + 1);
                TypeBuilder tb = this.module.DefineType(typeName, TypeAttributes.Class |
                    TypeAttributes.Public, typeof(DynamicClass));
                System.Reflection.FieldInfo[] fields = GenerateProperties(tb, properties);
                Type result = tb.CreateType();
                classCount++;
                return result;
            }
            finally
            {
                rwLock.DowngradeFromWriterLock(ref cookie);
            }
        }

        System.Reflection.FieldInfo[] GenerateProperties(TypeBuilder tb, DynamicProperty[]
properties)
        {
            System.Reflection.FieldInfo[] fields = new FieldBuilder[properties.Length];
            for (int i = 0; i < properties.Length; i++)
            {
                DynamicProperty dp = properties[i];
                FieldBuilder fb = tb.DefineField("_" + dp.Name, dp.Type,
FieldAttributes.Private);
```

```
                PropertyBuilder pb = tb.DefineProperty(dp.Name, PropertyAttributes.HasDefault,
dp.Type, null);
                MethodBuilder mbGet = tb.DefineMethod("get_" + dp.Name,
                    MethodAttributes.Public | MethodAttributes.SpecialName |
MethodAttributes.HideBySig,
                    dp.Type, Type.EmptyTypes);
                ILGenerator genGet = mbGet.GetILGenerator();
                genGet.Emit(OpCodes.Ldarg_0);
                genGet.Emit(OpCodes.Ldfld, fb);
                genGet.Emit(OpCodes.Ret);
                MethodBuilder mbSet = tb.DefineMethod("set_" + dp.Name,
                    MethodAttributes.Public | MethodAttributes.SpecialName |
MethodAttributes.HideBySig,
                    null, new Type[] { dp.Type });
                ILGenerator genSet = mbSet.GetILGenerator();
                genSet.Emit(OpCodes.Ldarg_0);
                genSet.Emit(OpCodes.Ldarg_1);
                genSet.Emit(OpCodes.Stfld, fb);
                genSet.Emit(OpCodes.Ret);
                pb.SetGetMethod(mbGet);
                pb.SetSetMethod(mbSet);
                fields[i] = fb;
            }
            return fields;
        }

        public static Type CreateClass(IEnumerable<DynamicProperty> properties)
        {
            return ClassFactory.Instance.GetDynamicClass(properties);
        }
    }
}
```

Файл A2v10.Designer.cpp

```
// Copyright © 2015-2018 Alex Kukhtin. All rights reserved.


#include "stdafx.h"
#include "A2v10.Designer.h"
#include "mainfrm.h"

#include "childfrm.h"

#include "moduledoc.h"
#include "sciview.h"
#include "moduleview.h"
#include "a2formdoc.h"
#include "a2formview.h"
#include "a2formtab.h"


#ifdef _DEBUG
#define new DEBUG_NEW
#endif

#pragma comment(lib,"../../bin/A2v10.Base.lib")
#pragma comment(lib,"../../bin/A2v10.Net.Shim.lib")

// CMainApp

BEGIN_MESSAGE_MAP(CMainApp, CA2WinApp)
	// Standard file based document commands
	ON_COMMAND(ID_FILE_NEW, OnFileNew)
	ON_COMMAND(ID_FILE_OPEN, OnFileOpen)
	// Standard print setup command
	ON_COMMAND(ID_FILE_PRINT_SETUP, OnFilePrintSetup)
END_MESSAGE_MAP()


// CMainApp construction

CMainApp::CMainApp()
	: CA2WinApp()
{
	// support Restart Manager
	m_dwRestartManagerSupportFlags = AFX_RESTART_MANAGER_SUPPORT_ALL_ASPECTS;
#ifdef _MANAGED
	// If the application is built using Common Language Runtime support (/clr):
	//     1) This additional setting is needed for Restart Manager support to work
properly.
	//     2) In your project, you must add a reference to System.Windows.Forms in order to
build.
	System::Windows::Forms::Application::SetUnhandledExceptionMode(System::Windows::Forms::
UnhandledExceptionMode::ThrowException);
#endif

	// TODO: replace application ID string below with unique ID string; recommended
	// format for string is CompanyName.ProductName.SubProduct.VersionInformation
	SetAppID(_T("A2v10.Designer.AppID.NoVersion"));

}

// The one and only CMainApp object

CMainApp theApp;
```

```
// CMainApp initialization

BOOL CMainApp::InitInstance()
{
	if (!__super::InitInstance())
		return FALSE;

	// Register the application's document templates.  Document templates
	//  serve as the connection between documents, frame windows and views
	try
	{
		CA2DocTemplate* pModuleTemplate = new CA2DocTemplate(IDR_JSMODULE,
			RUNTIME_CLASS(CModuleDoc),
			RUNTIME_CLASS(CChildFrame), // custom MDI child frame
			RUNTIME_CLASS(CModuleView));
		AddDocTemplate(pModuleTemplate);

		CA2DocTemplate* pFormTemplate = new CA2DocTemplate(IDR_FORM,
			RUNTIME_CLASS(CA2FormDocument),
			RUNTIME_CLASS(CChildFrame), //
			RUNTIME_CLASS(CA2FormTabView));
		AddDocTemplate(pFormTemplate);
	}
	catch (std::bad_alloc&) {
		return FALSE;
	}

	// create main MDI Frame window
	CMainFrame* pMainFrame = new CMainFrame;
	if (!pMainFrame || !pMainFrame->LoadFrame(IDR_MAINFRAME))
	{
		delete pMainFrame;
		return FALSE;
	}
	m_pMainWnd = pMainFrame;


	// Parse command line for standard shell commands, DDE, file open
	CA2CommandLineInfo cmdInfo;
	ParseCommandLine(cmdInfo);

	CAppData::SetDebug(cmdInfo.IsDebugMode());

	// Dispatch commands specified on the command line.  Will return FALSE if
	// app was launched with /RegServer, /Register, /Unregserver or /Unregister.
	//if (!ProcessShellCommand(cmdInfo))
		//return FALSE;

	try
	{
		CDotNetRuntime::Start();
		JavaScriptRuntime::CreateGlobalObject();
		CDotNetRuntime::LoadLibrary();
		JavaScriptRuntime::StartDebugging();
	}
	catch (CDotNetException& de)
	{
		de.ReportError();
		return FALSE;
	}

	// The main window has been initialized, so show and update it
	pMainFrame->ShowWindow(m_nCmdShow);
```

```
	pMainFrame->UpdateWindow();

	return TRUE;
}

int CMainApp::ExitInstance()
{
	try
	{
		CDotNetRuntime::Stop();
	}
	catch (CDotNetException& /*de*/)
	{
		// do nothing
	}

	return __super::ExitInstance();
}

// CMainApp message handlers
```

Файл a2mdiframe.cpp

```
// Copyright © 2008-2017 Alex Kukhtin. All rights reserved.

#include "stdafx.h"

#include "..\Include\appdefs.h"
#include "..\Include\a2glowborder.h"
#include "..\Include\a2captionbutton.h"
#include "..\Include\a2borderpane.h"
#include "..\Include\a2mdiframe.h"
#include "..\include\a2visualmanager.h"
#include "..\include\guiext.h"
#include "..\include\theme.h"
#include "..\include\uitools.h"
#include "Resource.h"


#pragma comment(lib, "dwmapi")

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CA2MDIFrameWnd

IMPLEMENT_DYNCREATE(CA2MDIFrameWnd, CMDIFrameWndEx)

CA2MDIFrameWnd::CA2MDIFrameWnd()
	: m_nDelta8(8), m_dwIdleFlags(0)
{
	CMFCVisualManagerOffice2003::SetUseGlobalTheme(FALSE);
	CMFCVisualManagerOffice2003::SetDefaultWinXPColors(FALSE);
}

CA2MDIFrameWnd::~CA2MDIFrameWnd()
{
}


BEGIN_MESSAGE_MAP(CA2MDIFrameWnd, CMDIFrameWndEx)
	ON_MESSAGE(WM_NCHITTEST, OnNcHitTest)
	ON_WM_PAINT()
	ON_WM_NCMOUSEMOVE()
	ON_MESSAGE(WM_SETMESSAGESTRING, OnSetMessageString)
	ON_MESSAGE(WM_IDLEUPDATECMDUI, OnIdleUpdateCmdUI)
	ON_MESSAGE(WM_NCCALCSIZE, OnNcCalcSize)
	ON_WM_NCMOUSELEAVE()
	ON_WM_ERASEBKGND()
	ON_WM_NCRBUTTONUP()
	ON_WM_NCLBUTTONDOWN()
	ON_WM_CREATE()
	ON_WM_WINDOWPOSCHANGED()
	ON_MESSAGE(WMI_IDLE_UPDATE, OnIdleUpdate)
	ON_REGISTERED_MESSAGE(AFX_WM_ON_MOVETOTABGROUP, OnMoveToTabGroup)
	ON_WM_SETTINGCHANGE()
	ON_MESSAGE(WMI_DEBUG_MODE, OnChangeDebugMode)
	ON_COMMAND(ID_WINDOW_MANAGER, OnWindowManager)
END_MESSAGE_MAP()


BOOL CA2MDIFrameWnd::CreateBorderPanes()
{
	return m_borderPanes.Create(this);
```

```
}
void CA2MDIFrameWnd::DockBorderPanes()
{
	m_borderPanes.DockPanes(this);
}

void CA2MDIFrameWnd::EnableDefaultMDITabbedGroups()
{
	CMDITabInfo mdiTabParams;
	mdiTabParams.m_style = CMFCTabCtrl::STYLE_3D_SCROLLED; // other styles available...
	mdiTabParams.m_bTabCloseButton = TRUE;
	mdiTabParams.m_bActiveTabCloseButton = TRUE;   // set to FALSE to place close button at
right of tab area
	mdiTabParams.m_bTabIcons = TRUE;    // set to TRUE to enable document icons on MDI taba
	mdiTabParams.m_bAutoColor = FALSE;    // set to FALSE to disable auto-coloring of MDI
tabs
	mdiTabParams.m_bDocumentMenu = TRUE; // enable the document menu at the right edge of
the tab area
	mdiTabParams.m_nTabBorderSize = 1; // нужно, чтобы правильно нарисовать рамки
	mdiTabParams.m_bFlatFrame = TRUE;
	mdiTabParams.m_bReuseRemovedTabGroups = FALSE;
	EnableMDITabbedGroups(TRUE, mdiTabParams);
}

int CA2MDIFrameWnd::GetCaptionHeight()
{
	return max(::GetSystemMetrics(SM_CYCAPTION) + 4, 28);
}

void CA2MDIFrameWnd::UpdateTabs()
{
	m_wndClientArea.UpdateTabs();
}

void CA2MDIFrameWnd::UpdateMdiTabs()
{
	// MFC BUG. Update artifacts
	const CObList& obList = m_wndClientArea.GetMDITabGroups();
	POSITION pos = obList.GetHeadPosition();
	while (pos) {
		CMFCTabCtrl* pTab = DYNAMIC_DOWNCAST(CMFCTabCtrl, obList.GetNext(pos));
		pTab->RecalcLayout();
		pTab->EnsureVisible(pTab->GetActiveTab());
	}
}

// afx_msg
int CA2MDIFrameWnd::OnCreate(LPCREATESTRUCT lpCreateStruct)
{

	if (__super::OnCreate(lpCreateStruct) == -1)
		return -1;

	if (!m_glowBorder.Create(this))
		return -1;

	CMFCMenuBar::EnableMenuShadows(FALSE);
	ModifyStyle(0, WS_CLIPCHILDREN, 0);
	ModifyStyleEx(WS_EX_CLIENTEDGE, WS_EX_APPWINDOW | WS_EX_WINDOWEDGE);

	MARGINS margins = { 0, 0, 0, 0 };
	HRESULT hr = ::DwmExtendFrameIntoClientArea(GetSafeHwnd(), &margins);
	if (!SUCCEEDED(hr)) {
		//ATLASSERT(FALSE);
```

```
	}

	return 0;
}

// virtual
void CA2MDIFrameWnd::GetMessageString(UINT nID, CString& rMessage) const
{
	if (nID == 0)
		return;
	__super::GetMessageString(nID, rMessage); // needed for tooltip
}

// virtual
void CA2MDIFrameWnd::AdjustDockingLayout(HDWP hdwp /*= NULL*/)
{
	__super::AdjustDockingLayout(hdwp);
	RecalcLayout(); // always
}

// virtual
void CA2MDIFrameWnd::RecalcLayout(BOOL bNotify /*= TRUE*/)
{
	if (m_bInRecalcLayout)
		return;
	CRect winRect(0, 0, 200, 200);
	AdjustWindowRectEx(winRect, WS_OVERLAPPEDWINDOW, FALSE, WS_EX_APPWINDOW);
	m_nDelta8 = (winRect.Width() - 200) / 2;
	BOOL bZoomed = IsZoomed(); // GetStyle() & WS_MAXIMIZE;
	CRect clientRect;
	GetClientRect(clientRect);
	int cyCaption = GetCaptionHeight();
	CRect captionRect = clientRect;
	captionRect.bottom = captionRect.top + cyCaption;
	if (bZoomed) {
		m_dockManager.m_rectInPlace = clientRect;
		m_dockManager.m_rectInPlace.DeflateRect(m_nDelta8, cyCaption + m_nDelta8,
m_nDelta8, m_nDelta8);
		captionRect.OffsetRect(-m_nDelta8, m_nDelta8);
		m_captionButtons.RecalcLayout(captionRect, bZoomed);
		__super::RecalcLayout(bNotify);
		return;
	}
	m_dockManager.m_rectInPlace = clientRect;
	// caption only!!!
	m_dockManager.m_rectInPlace.DeflateRect(0, cyCaption, 0, 0);
	m_captionButtons.RecalcLayout(captionRect, bZoomed);
	__super::RecalcLayout(bNotify);

	// MFC BUG. Update artifacts
	const CObList& obList = m_wndClientArea.GetMDITabGroups();
	POSITION pos = obList.GetHeadPosition();
	while (pos) {
		CMFCTabCtrl* pTab = DYNAMIC_DOWNCAST(CMFCTabCtrl, obList.GetNext(pos));
		pTab->Invalidate();
	}
}

BOOL CA2MDIFrameWnd::PreTranslateMessage(MSG* pMsg)
{
	//MFC:hack disable activate menu on F1
	if (pMsg->message == WM_SYSKEYUP && pMsg->wParam == VK_F10) {
		return TRUE;
	}
```

```
	return __super::PreTranslateMessage(pMsg);
}

// afx_msg
void CA2MDIFrameWnd::OnWindowPosChanged(WINDOWPOS* lpwndpos)
{
	__super::OnWindowPosChanged(lpwndpos);
	m_glowBorder.OnWindowPosChanged(this);
	Invalidate();
}

// afx_msg
LRESULT CA2MDIFrameWnd::OnNcCalcSize(WPARAM wParam, LPARAM lParam)
{
	return 0;
}

// afx_msg
LRESULT CA2MDIFrameWnd::OnMoveToTabGroup(WPARAM, LPARAM)
{
	// MFC BUG. Update artifacts
	RecalcLayout();
	UpdateMdiTabs();
	return 0L;
}

// afx_msg
LRESULT CA2MDIFrameWnd::OnNcHitTest(WPARAM wParam, LPARAM lParam)
{
	CPoint pt(lParam);
	CRect wr;
	GetWindowRect(wr);
	int cyCaption = GetCaptionHeight();
	int dxIcon = (cyCaption - 24) / 2;
	if (pt.y < (wr.top + cyCaption)) {
		if (pt.x < (wr.left + cyCaption + dxIcon * 2))
			return HTSYSMENU;
		else if (pt.x > wr.right - m_captionButtons.Width())
			return HTOBJECT;
		else
			return HTCAPTION;
	}
	return HTNOWHERE;
}

// afx_msg
void CA2MDIFrameWnd::OnNcRButtonUp(UINT nHitTest, CPoint point)
{
	if (nHitTest != HTCAPTION)
		return;
	CMenu* pMenu = GetSystemMenu(FALSE);
	if (pMenu->GetSafeHmenu() != NULL && ::IsMenu(pMenu->GetSafeHmenu()))
	{
		pMenu->EnableMenuItem(SC_MAXIMIZE, MF_BYCOMMAND | MF_ENABLED);
		pMenu->EnableMenuItem(SC_RESTORE, MF_BYCOMMAND | MF_ENABLED);

		if (IsZoomed())
		{
			pMenu->EnableMenuItem(SC_MAXIMIZE, MF_BYCOMMAND | MF_DISABLED |
MF_GRAYED);
		}
		else if (!IsIconic())
		{
```

```
                        pMenu->EnableMenuItem(SC_RESTORE, MF_BYCOMMAND | MF_DISABLED |
MF_GRAYED);
                }

                UINT uiRes = ::TrackPopupMenu(pMenu->GetSafeHmenu(), TPM_LEFTBUTTON |
TPM_RETURNCMD, point.x, point.y, 0, GetSafeHwnd(), NULL);
                if (uiRes != 0)
                {
                        PostMessage(WM_SYSCOMMAND, uiRes);
                }
        }
}

// afx_msg
void CA2MDIFrameWnd::OnNcLButtonDown(UINT nHitTest, CPoint point)
{
        __super::OnNcLButtonDown(nHitTest, point);
        if (nHitTest == HTOBJECT) {
                ScreenToClient(&point);
                m_captionButtons.PressButton(point, this);
        }
}


// afx_msg
void CA2MDIFrameWnd::OnNcMouseMove(UINT nHitTest, CPoint point)
{
        __super::OnNcMouseMove(nHitTest, point);
        if (nHitTest == HTOBJECT) {
                ScreenToClient(&point);
                if (m_captionButtons.MouseMove(point))
                        InvalidateRect(m_captionButtons.GetRect());
        }
        else {
                if (m_captionButtons.ClearHighlight())
                        InvalidateRect(m_captionButtons.GetRect());
        }
}

// afx_msg
void CA2MDIFrameWnd::OnNcMouseLeave()
{
        __super::OnNcMouseLeave();
        if (m_captionButtons.ClearHighlight())
                InvalidateRect(m_captionButtons.GetRect());
}


// afx_msg
void CA2MDIFrameWnd::OnPaint()
{
        CPaintDC dc(this);
        CRect rc;
        GetClientRect(rc);

        BOOL bMax = IsZoomed();

        int cyCaption = GetCaptionHeight();
        int dxIcon = (cyCaption - 24) / 2;
        int iconOrigin = 0;

        CRect captionRect(rc.left, rc.top, rc.right, rc.top + cyCaption);
        if (bMax) {
                captionRect.OffsetRect(m_nDelta8, m_nDelta8);
```

```
		iconOrigin = m_nDelta8;
	}

	auto pVm = DYNAMIC_DOWNCAST(CA2VisualManager, CMFCVisualManager::GetInstance());
	if (pVm)
		dc.FillRect(captionRect, pVm->GetWindowCaptionBackgroundBrush()); // menu bar
background

					//dc.Draw3dRect(captionRect, RGB(255, 255, 0), RGB(255,
255, 0));

	dc.SetBkMode(TRANSPARENT);
	CFont* pOldFont = dc.SelectObject(CTheme::GetUIFont(CTheme::FontNonClient));
	captionRect.left += cyCaption + dxIcon * 2;
	CString str;
	GetWindowText(str);
	dc.SetTextColor(RGB(0x33, 0x33, 0x33)); /***???**/
	captionRect.right -= m_captionButtons.Width();
	dc.DrawText(str, captionRect, DT_LEFT | DT_VCENTER | DT_SINGLELINE | DT_END_ELLIPSIS);
	dc.SelectObject(pOldFont);
	static HICON hIcon = NULL;
	if (hIcon == NULL) {
		hIcon = (HICON) ::LoadImage(AfxGetInstanceHandle(),
MAKEINTRESOURCE(IDR_MAINFRAME), IMAGE_ICON, 24, 24, 0);
	}
	::DrawIconEx(dc.GetSafeHdc(), iconOrigin + dxIcon * 2, iconOrigin + dxIcon, hIcon, 24,
24, 0, NULL, DI_NORMAL);
	m_captionButtons.Draw(&dc);
}

// afx_msg
LRESULT CA2MDIFrameWnd::OnIdleUpdate(WPARAM wParam, LPARAM lParam)
{
	if (wParam == WMI_IDLE_UPDATE_WPARAM)
		m_dwIdleFlags |= lParam;
	return 0L;
}

// afx_msg
LRESULT CA2MDIFrameWnd::OnIdleUpdateCmdUI(WPARAM, LPARAM)
{
	m_dockManager.SendMessageToMiniFrames(WM_IDLEUPDATECMDUI);
	if (m_dwIdleFlags & IDLE_UPDATE_MDITABS) {
		UpdateMdiTabs();
		m_dwIdleFlags &= ~IDLE_UPDATE_MDITABS;
	}
	return 0L;
}

// afx_msg
BOOL CA2MDIFrameWnd::OnEraseBkgnd(CDC* pDC)
{
	return TRUE;
}

// virtual
void CA2MDIFrameWnd::OnDebugModeChanged(bool bDebug)
{

}

// afx_msg
LRESULT CA2MDIFrameWnd::OnSetMessageString(WPARAM wParam, LPARAM lParam)
{
```

```
	UINT nIDLast = m_nIDLastMessage;
	m_nIDLastMessage = (UINT)wParam;    // new ID (or 0)
	m_nIDTracking = (UINT)wParam;       // so F1 on toolbar buttons work
	return nIDLast;
}


// virtual
BOOL CA2MDIFrameWnd::OnCmdMsg(UINT nID, int nCode, void* pExtra, AFX_CMDHANDLERINFO*
pHandlerInfo)
{
	if (CUITools::TryDoCmdMsg(nID, nCode, pExtra, pHandlerInfo))
		return TRUE;
	return __super::OnCmdMsg(nID, nCode, pExtra, pHandlerInfo);
}

void CA2MDIFrameWnd::OnSettingChange(UINT uFlags, LPCTSTR lpszSection)
{
	__super::OnSettingChange(uFlags, lpszSection);
	CTheme::OnSettingChange();
	SendMessageToDescendants(WMI_SETTINGCHANGE, WPARAM(uFlags), (LPARAM)lpszSection, TRUE,
TRUE);
}

// afx_msg
LRESULT CA2MDIFrameWnd::OnChangeDebugMode(WPARAM wParam, LPARAM lParam)
{
	if (wParam != WMI_DEBUG_MODE_WPARAM)
		return 0L;
	auto pVm = DYNAMIC_DOWNCAST(CA2VisualManager, CMFCVisualManager::GetInstance());
	bool bMode = lParam ? true : false;
	if (pVm->SetDebugMode(bMode))
		OnDebugModeChanged(bMode);
	return 0L;
}

void CA2MDIFrameWnd::OnWindowManager()
{
	ShowWindowsDialog();
}
```

Файл javascriptruntime.cpp

```
#include "stdafx.h"

#include "../include/javascriptpropertyid.h"
#include "../include/javascriptvalue.h"
#include "../include/javascriptruntime.h"
#include "../include/javascriptnative.h"
#include "../include/javascriptexceptions.h"

#include "../include/appdefs.h"
#include "../include/filetools.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

JsRuntimeHandle s_runtime = JS_INVALID_RUNTIME_HANDLE;
volatile bool s_bInDebugMode = false;
volatile bool s_bClosingProgress = false;

JsSourceContext s_currentContext = JS_SOURCE_CONTEXT_NONE;

// static
JsRuntimeHandle JavaScriptRuntime::CurrentRuntime()
{
	if (s_runtime != JS_INVALID_RUNTIME_HANDLE)
		return s_runtime;
	JsContextRef context = JS_INVALID_REFERENCE;
	JavaScriptNative::ThrowIfError(JsGetCurrentContext(&context));
	JavaScriptNative::ThrowIfError(JsGetRuntime(context, &s_runtime));
	return s_runtime;
}

JsValueRef CHAKRA_CALLBACK RequireCallback(_In_ JsValueRef callee, _In_ bool isConstructCall,
_In_ JsValueRef *arguments, _In_ unsigned short argumentCount, _In_opt_ void *callbackState)
{
	CString msg;
	try
	{
		if (argumentCount < 3)
			throw JavaScriptUsageException(JsErrorCode::JsErrorInvalidArgument,
L"__require");
		CString fileName = JavaScriptValue(arguments[1]).ConvertToString().ToString();
		CString pathName = JavaScriptValue(arguments[2]).ConvertToString().ToString();
		WCHAR fullPath[_MAX_PATH + 1];
		::PathCombine(fullPath, pathName, fileName);
		::PathAddExtension(fullPath, L".js");
		if (!::PathFileExists(fullPath)) {
			msg.Format(L"File '%s' not found", fullPath);
			throw JavaScriptUsageException(JsErrorCode::JsErrorInvalidArgument, msg);
		}
		CString code;
		if (!CFileTools::LoadFile(fullPath, code)) {
			msg.Format(L"Error reading from file '%s'", fullPath);
			throw JavaScriptUsageException(JsErrorCode::JsErrorScriptException, msg);
		}
		return JavaScriptRuntime::RunModule(code, fullPath);
	}
	catch (JavaScriptException& jsEx)
	{
		jsEx.SetException();
	}
	catch (...)
```

```
	{
		JavaScriptRuntime::SetUnknownException();
	}
	return JS_INVALID_REFERENCE;
}

JsValueRef CHAKRA_CALLBACK AlertCallback(_In_ JsValueRef callee, _In_ bool isConstructCall,
_In_ JsValueRef *arguments, _In_ unsigned short argumentCount, _In_opt_ void *callbackState)
{
	try {
		CString strMessage(EMPTYSTR);
		if (argumentCount > 1)
			strMessage = JavaScriptValue(arguments[1]).ConvertToString().ToString();
		AfxMessageBox(strMessage);
	}
	catch (JavaScriptException& jsEx)
	{
		jsEx.SetException();
	}
	catch (...)
	{
		JavaScriptRuntime::SetUnknownException();
	}
	return JS_INVALID_REFERENCE;
}

JsValueRef CHAKRA_CALLBACK LogCallback(_In_ JsValueRef callee, _In_ bool isConstructCall, _In_
JsValueRef *arguments, _In_ unsigned short argumentCount, _In_opt_ void *callbackState)
{
	try
	{
		CString strMessage(EMPTYSTR);
		if (argumentCount > 1)
			strMessage = JavaScriptValue(arguments[1]).ConvertToString().ToString();
		WPARAM wParam = reinterpret_cast<WPARAM>(callbackState);
		CWnd* pWnd = AfxGetMainWnd();
		if (pWnd) {
			pWnd->SendMessage(WMI_CONSOLE, wParam, (LPARAM)(LPCWSTR)strMessage);
		}
	}
	catch (JavaScriptException& jsEx)
	{
		jsEx.SetException();
	}
	catch (...)
	{
		JavaScriptRuntime::SetUnknownException();
	}
	return JS_INVALID_REFERENCE;
}

// static
void JavaScriptRuntime::CreateGlobalObject()
{
	// in CURRENT (global) context
	auto glob = JavaScriptValue::GlobalObject();

	auto alert = JavaScriptValue::CreateFunction(AlertCallback, nullptr);
	glob.SetProperty(L"alert", alert);

	auto console = JavaScriptValue::CreateObject();
	glob.SetProperty(L"console", console);

	auto log = JavaScriptValue::CreateFunction(LogCallback, (void*) WMI_CONSOLE_LOG);
```

```
        console.SetProperty(L"log", log);

        auto require = JavaScriptValue::CreateFunction(RequireCallback, nullptr);
        glob.SetProperty(L"__require", require);

}

// static
CString JavaScriptRuntime::Evaluate(const wchar_t* szScript)
{
        JavaScriptValue result;
        JavaScriptNative::ThrowIfError(JsRunScript(szScript, JS_SOURCE_CONTEXT_NONE, L"",
result));
        return result.ConvertToString().ToString();
}

bool JavaScriptRuntime::RunScript(LPCWSTR szCode, LPCWSTR szPathName)
{
        JavaScriptValue result = JS_INVALID_REFERENCE;
        s_currentContext += 1;
        int context = s_currentContext;
        JavaScriptNative::ThrowIfError(JsRunScript(szCode, context, szPathName, result));
        return s_bClosingProgress;
}

JavaScriptValue JavaScriptRuntime::RunModule(LPCWSTR szCode, LPCWSTR szPathName)
{
        JavaScriptValue result = JS_INVALID_REFERENCE;
        s_currentContext += 1;
        int context = s_currentContext;
        LPCWSTR szPrologue = L"(function() { let m = {exports: {}}; (function(module, exports)
{ ";
        LPCWSTR szEpilogue = L" })(m, m.exports); return m.exports;})();";
        CString codeToRun(szPrologue);
        codeToRun += szCode;
        codeToRun += szEpilogue;
        JavaScriptNative::ThrowIfError(JsRunScript(codeToRun, context, szPathName, result));
        return result;
}

// static
void JavaScriptRuntime::SetException(JavaScriptValue exception)
{
        JavaScriptNative::ThrowIfError(JsSetException(exception));
}

// static
void JavaScriptRuntime::SetUnknownException()
{
        auto err = JavaScriptValue::CreateError(JavaScriptValue::FromString(L"Unknown error"));
        JavaScriptRuntime::SetException(err);
}


// static
JsContextRef JavaScriptRuntime::CreateContext()
{
        JsContextRef newContext = JS_INVALID_REFERENCE;
        JavaScriptNative::ThrowIfError(JsCreateContext(CurrentRuntime(), &newContext));
        return newContext;
}

// static
JavaScriptValue JavaScriptRuntime::CreateDesignerElement(const wchar_t* szJson)
```

```
{
	auto createElem =
JavaScriptValue::GlobalObject().GetPropertyChain(L"designer.form.__createElement");
	return createElem.CallFunctionArg(JavaScriptValue::FromString(szJson));
}

static int processEvents()
{
	CWinThread* pThis = AfxGetApp();
	ASSERT_VALID(pThis);

	_AFX_THREAD_STATE* pState = AfxGetThreadState();

	// for tracking the idle time state
	BOOL bIdle = TRUE;
	LONG lIdleCount = 0;

	// acquire and dispatch messages until a WM_QUIT message is received.
	for (;;)
	{
		// phase1: check to see if we can do idle work
		while (bIdle &&
			!::PeekMessage(&(pState->m_msgCur), NULL, NULL, NULL, PM_NOREMOVE))
		{
			// call OnIdle while in bIdle state
			if (!pThis->OnIdle(lIdleCount++))
				bIdle = FALSE; // assume "no idle" state
		}

		// phase2: pump messages while available
		do
		{
			// pump message, but quit on WM_QUIT
			if (!pThis->PumpMessage()) {
				// simply complete DiagDebugEventCallback and send quit message
again
				JavaScriptRuntime::SetDebugMode(false);
				s_bClosingProgress = true;
				PostQuitMessage(0);
				return 0;
			}
			// reset "no idle" state after pumping "normal" message
			//if (IsIdleMessage(&m_msgCur))
			if (pThis->IsIdleMessage(&(pState->m_msgCur)))
			{
				bIdle = TRUE;
				lIdleCount = 0;
			}
			if (!JavaScriptRuntime::InDebugMode()) {
				JavaScriptRuntime::ExitDebugMode();
				return 0;
			}

		} while (::PeekMessage(&(pState->m_msgCur), NULL, NULL, NULL, PM_NOREMOVE));
	}
}

static void _sendDebugInfo(JsValueRef eventData)
{
	if (s_bClosingProgress)
		return;
	JavaScriptValue eventInfo(eventData);

	int lineNo = eventInfo.GetProperty(L"line").ToInt();
```

```
	int scriptId = eventInfo.GetProperty(L"scriptId").ToInt();

	CString fileName = JavaScriptRuntime::GetFileNameFromScriptId(scriptId);

	DEBUG_BREAK_INFO breakInfo;
	breakInfo.szFileName = (LPCWSTR) fileName;
	breakInfo.scriptId = scriptId;
	breakInfo.lineNo = lineNo;
	AfxGetMainWnd()->SendMessage(WMI_DEBUG_BREAK, WMI_DEBUG_BREAK_WPARAM,
(LPARAM)&breakInfo);
}

CString JavaScriptRuntime::GetFileNameFromScriptId(int scriptId)
{
	JavaScriptValue arr;
	JavaScriptNative::ThrowIfError(JsDiagGetScripts(arr));
	int len = arr.GetProperty(L"length").ToInt();
	auto fileNamePropId = JavaScriptPropertyId::FromString(L"fileName");
	auto scriptIdPropId = JavaScriptPropertyId::FromString(L"scriptId");
	for (int i = 0; i < len; i++) {
		JavaScriptValue item = arr.GetProperty(i);
		int itemId = item.GetProperty(scriptIdPropId).ToInt();
		if (itemId == scriptId) {
			JavaScriptValue fileNameVal = item.GetProperty(fileNamePropId);
			if (fileNameVal.ValueType() == JsString) {
				return fileNameVal.ToString();
			}
		}
	}
	return L"";
}

void CHAKRA_CALLBACK DiagDebugEventCallback(_In_ JsDiagDebugEvent debugEvent, _In_ JsValueRef
eventData, _In_opt_ void* callbackState)
{
	if (s_bClosingProgress)
		return;
	if ((debugEvent == JsDiagDebugEvent::JsDiagDebugEventDebuggerStatement) ||
		(debugEvent == JsDiagDebugEvent::JsDiagDebugEventStepComplete) ||
		(debugEvent == JsDiagDebugEvent::JsDiagDebugEventBreakpoint))
	{
		JavaScriptRuntime::SetDebugMode(true);
		JavaScriptRuntime::EnterDebugMode();
		_sendDebugInfo(eventData);
		//auto str =
JavaScriptValue::GlobalObject().GetPropertyChain(L"JSON.stringify");
		//auto data = str.CallFunction(JavaScriptValue::Undefined(),
eventData).ToString();
		processEvents();
	}
}

void JavaScriptRuntime::StartDebugging()
{
	JavaScriptNative::ThrowIfError(JsDiagStartDebugging(CurrentRuntime(),
DiagDebugEventCallback, nullptr));
}

void JavaScriptRuntime::StopDebugging()
{
	void* pState = nullptr;
	JavaScriptNative::ThrowIfError(JsDiagStopDebugging(CurrentRuntime(), &pState));
}
```

```
// static
bool JavaScriptRuntime::InDebugMode()
{
	return s_bInDebugMode;
}

void JavaScriptRuntime::SetDebugMode(bool bSet)
{
	if (s_bInDebugMode == bSet)
		return;
	s_bInDebugMode = bSet;
	if (bSet)
		AfxGetMainWnd()->PostMessage(WMI_DEBUG_MODE, WMI_DEBUG_MODE_WPARAM,
(LPARAM)TRUE);
}

// static
void JavaScriptRuntime::EndRunScript()
{
	AfxGetMainWnd()->PostMessage(WMI_DEBUG_MODE, WMI_DEBUG_MODE_WPARAM, (LPARAM)FALSE);
}

void JavaScriptRuntime::ExitDebugMode()
{
}

// static
void JavaScriptRuntime::EnterDebugMode()
{
}

// static
void JavaScriptRuntime::SetDebugStepType(DebugStepType step)
{
	JsDiagStepType dt = JsDiagStepTypeContinue;
	switch (step) {
	case StepIn: dt = JsDiagStepTypeStepIn; break;
	case StepOut: dt = JsDiagStepTypeStepOut; break;
	case StepOver: dt = JsDiagStepTypeStepOver; break;
	case Continue: dt = JsDiagStepTypeContinue; break;
	default:
		ATLASSERT(FALSE);
	}
	JavaScriptNative::ThrowIfError(JsDiagSetStepType(dt));
}

JavaScriptContext::JavaScriptContext()
{
	JavaScriptNative::ThrowIfError(JsGetCurrentContext(&m_prevContext));
	JsContextRef newContext = JavaScriptRuntime::CreateContext();

	auto globSrc = JavaScriptValue::GlobalObject();
	JavaScriptNative::ThrowIfError(JsSetCurrentContext(newContext));
	auto globTrg = JavaScriptValue::GlobalObject();

	auto alertPropId = JavaScriptPropertyId::FromString(L"alert");
	auto consolePropId = JavaScriptPropertyId::FromString(L"console");
	auto requirePropId = JavaScriptPropertyId::FromString(L"__require");
	auto alertVal = globTrg.GetProperty(alertPropId);
	if (alertVal.ValueType() != JsUndefined)
		return; // already set
	alertVal = globSrc.GetProperty(alertPropId);
	globTrg.SetProperty(alertPropId, alertVal);
	auto consoleVal = globSrc.GetProperty(consolePropId);
```

```
        globTrg.SetProperty(consolePropId, consoleVal);
        auto requireVal = globSrc.GetProperty(requirePropId);
        globTrg.SetProperty(requirePropId, requireVal);
        //TODO: parse app objects in this context (app, require, etc)
}

JavaScriptContext::~JavaScriptContext()
{
        JavaScriptNative::ThrowIfError(JsSetCurrentContext(m_prevContext));
}
```

Файл BaseController.cs

```
// Copyright © 2015-2017 Alex Kukhtin. All rights reserved.

using System;
using System.Dynamic;
using System.IO;
using System.Text;
using System.Threading.Tasks;
using Newtonsoft.Json;

using A2v10.Infrastructure;
using System.Collections.Generic;
using A2v10.Infrastructure.Utilities;
using System.Net;
using A2v10.Data.Interfaces;

namespace A2v10.Request
{
    public partial class BaseController
    {
        protected IApplicationHost _host;
        protected IDbContext _dbContext;
        protected IRenderer _renderer;
        protected IWorkflowEngine _workflowEngine;
        protected ILocalizer _localizer;
        protected IDataScripter _scripter;

        public BaseController()
        {
            // DI ready
            IServiceLocator locator = ServiceLocator.Current;
            _host = locator.GetService<IApplicationHost>();
            _dbContext = locator.GetService<IDbContext>();
            _renderer = locator.GetService<IRenderer>();
            _workflowEngine = locator.GetService<IWorkflowEngine>();
            _localizer = locator.GetService<ILocalizer>();
            _scripter = locator.GetService<IDataScripter>();
        }

        public Boolean IsDebugConfiguration => _host.IsDebugConfiguration;
        public IDbContext DbContext => _dbContext;
        public IApplicationHost Host => _host;
        public Boolean Admin { get; set; }

        public String Localize(String content)
        {
            return _localizer.Localize(null, content);
        }

        public async Task RenderApplicationKind(RequestUrlKind kind, String pathInfo,
ExpandoObject loadPrms, TextWriter writer)
        {
            var segs = pathInfo.Split('/');
            if (segs.Length < 2)
                throw new RequestModelException($"Invalid application Url: {pathInfo}");
            if (segs[0] != "app")
                throw new RequestModelException($"Invalid application Url: {pathInfo}");
            switch (segs[1])
            {
                case "about":
                    if (kind != RequestUrlKind.Page)
                        throw new RequestModelException($"Invalid application Url:
{pathInfo}");
```

```
                    await RenderAbout(writer);
                    break;
                case "changepassword":
                    if (kind != RequestUrlKind.Dialog)
                        throw new RequestModelException($"Invalid application Url:
{pathInfo}");
                    await RenderChangePassword(writer);
                    break;
                default:
                    throw new RequestModelException($"Invalid application Url: {pathInfo}");
            }
        }

        public async Task RenderElementKind(RequestUrlKind kind, String pathInfo,
ExpandoObject loadPrms, TextWriter writer)
        {
            RequestModel rm = await RequestModel.CreateFromUrl(_host, Admin, kind, pathInfo);
            RequestView rw = rm.GetCurrentAction(kind);
            await Render(rw, writer, loadPrms);
        }

        async Task<RequestView> LoadIndirect(RequestView rw, IDataModel innerModel,
ExpandoObject loadPrms)
        {
            if (!rw.indirect)
                return rw;
            if (!String.IsNullOrEmpty(rw.target))
            {
                String targetUrl = innerModel.Root.Resolve(rw.target);
                if (String.IsNullOrEmpty(rw.targetId))
                    throw new RequestModelException("targetId must be specified for indirect
action");
                targetUrl += "/" + innerModel.Root.Resolve(rw.targetId);
                var rm = await RequestModel.CreateFromUrl(_host, Admin, rw.CurrentKind,
targetUrl);
                rw = rm.GetCurrentAction();
                String loadProc = rw.LoadProcedure;
                if (loadProc != null)
                {
                    loadPrms.Set("Id", rw.Id);
                    var newModel = await _dbContext.LoadModelAsync(rw.CurrentSource, loadProc,
loadPrms);
                    innerModel.Merge(newModel);
                    innerModel.System.Set("__indirectUrl__", rm.BaseUrl);
                }
            }
            else
            {
                // simple view/model redirect
                if (rw.targetModel == null)
                {
                    throw new RequestModelException("'targetModel' must be specified for
indirect action without 'target' property");

                }
                rw.model = innerModel.Root.Resolve(rw.targetModel.model);
                rw.view = innerModel.Root.Resolve(rw.targetModel.view);
                rw.schema = innerModel.Root.Resolve(rw.targetModel.schema);
                if (String.IsNullOrEmpty(rw.schema))
                    rw.schema = null;
                rw.template = innerModel.Root.Resolve(rw.targetModel.template);
                if (String.IsNullOrEmpty(rw.template))
                    rw.template = null;
                String loadProc = rw.LoadProcedure;
```

```
                if (loadProc != null)
                {
                    loadPrms.Set("Id", rw.Id);
                    var newModel = await _dbContext.LoadModelAsync(rw.CurrentSource, loadProc,
loadPrms);
                    innerModel.Merge(newModel);
                }
            }
            return rw;
        }

        protected async Task Render(RequestView rw, TextWriter writer, ExpandoObject loadPrms)
        {
            String loadProc = rw.LoadProcedure;
            IDataModel model = null;
            if (rw.parameters != null && loadPrms == null)
                loadPrms = rw.parameters;
            if (loadPrms != null)
            {
                loadPrms.Set("Id", rw.Id);
                loadPrms.Append(rw.parameters);
            }
            if (loadProc != null)
            {
                ExpandoObject prms2 = loadPrms;
                if (rw.indirect)
                {
                    // for indirect - @TenantId, @UserId and @Id only
                    prms2 = new ExpandoObject();
                    prms2.Set("Id", rw.Id);
                    if (loadPrms != null)
                    {
                        prms2.Set("UserId", loadPrms.Get<Int64>("UserId"));
                        prms2.Set("TenantId", loadPrms.Get<Int32>("TenantId"));
                    }
                }
                model = await _dbContext.LoadModelAsync(rw.CurrentSource, loadProc, prms2);
            }
            if (rw.indirect)
                rw = await LoadIndirect(rw, model, loadPrms);

            String viewName = rw.GetView();
            String rootId = "el" + Guid.NewGuid().ToString();

            String modelScript = await WriteModelScript(rw, model, rootId);

            // TODO: use view engines
            // try xaml
            String fileName = rw.GetView() + ".xaml";
            String filePath = _host.MakeFullPath(Admin, rw.Path, fileName);
            bool bRendered = false;
            if (System.IO.File.Exists(filePath))
            {
                // render XAML
                if (System.IO.File.Exists(filePath))
                {
                    using (var strWriter = new StringWriter())
                    {
                        var ri = new RenderInfo()
                        {
                            RootId = rootId,
                            FileName = filePath,
                            FileTitle = fileName,
                            Writer = strWriter,
```

```
                            DataModel = model,
                            Localizer = _localizer,
                            CurrentLocale = null
                        };
                        _renderer.Render(ri);
                        // write markup
                        writer.Write(strWriter.ToString());
                        bRendered = true;
                    }
                }
            }
            else
            {
                // try html
                fileName = rw.GetView() + ".html";
                filePath = _host.MakeFullPath(Admin, rw.Path, fileName);
                if (System.IO.File.Exists(filePath))
                {
                    using (_host.Profiler.CurrentRequest.Start(ProfileAction.Render, $"render:
{fileName}"))
                    {
                        using (var tr = new StreamReader(filePath))
                        {
                            String htmlText = await tr.ReadToEndAsync();
                            htmlText = htmlText.Replace("$(RootId)", rootId);
                            writer.Write(htmlText);
                            bRendered = true;
                        }
                    }
                }
            }
            if (!bRendered)
            {
                throw new RequestModelException($"The view '{rw.GetView()}' was not found. The
following locations were
searched:\n{rw.GetRelativePath(".xaml")}\n{rw.GetRelativePath(".html")}");
            }
            writer.Write(modelScript);
        }


        async Task<String> WriteModelScript(RequestView rw, IDataModel model, String rootId)
        {
            StringBuilder output = new StringBuilder();
            String dataModelText = "null";
            String templateText = "{}";
            StringBuilder sbRequired = new StringBuilder();
            if (model != null)
            {
                // write model script
                String fileTemplateText = null;
                if (rw.template != null)
                {
                    fileTemplateText = await _host.ReadTextFile(Admin, rw.Path, rw.template +
".js");
                    AddRequiredModules(sbRequired, fileTemplateText);
                    templateText = CreateTemplateForWrite(_localizer.Localize(null,
fileTemplateText));
                }
                dataModelText = JsonConvert.SerializeObject(model.Root,
StandardSerializerSettings);
            }
```

```
            const String scriptHeader =
@"
<script type=""text/javascript"">

'use strict';

$(RequiredModules)

(function() {
    const DataModelController = component('baseController');

    const rawData = $(DataModelText);
    const template = $(TemplateText);
";
            const String scriptFooter =
@"
    const vm = new DataModelController({
        el:'#$(RootId)',
        props: {
            inDialog: {type: Boolean, default: $(IsDialog)},
            pageTitle: {type: String}
        },
        data: modelData(template, rawData)
    });

    vm.$data._host_ = {
        $viewModel: vm
    };

    vm.__doInit__();

})();
</script>
";
            // TODO: may be data model from XAML ????
            const String emptyModel = "function modelData() {return null;}";

            var header = new StringBuilder(scriptHeader);
            header.Replace("$(RootId)", rootId);
            header.Replace("$(DataModelText)", dataModelText);
            header.Replace("$(TemplateText)", _localizer.Localize(null, templateText));
            header.Replace("$(RequiredModules)", sbRequired != null ? sbRequired.ToString() :
String.Empty);
            output.Append(header);
            if (model != null)
                output.Append(model.CreateScript(_scripter));
            else
                output.Append(emptyModel);
            var footer = new StringBuilder(scriptFooter);
            footer.Replace("$(RootId)", rootId);
            footer.Replace("$(IsDialog)", rw.IsDialog.ToString().ToLowerInvariant());
            output.Append(footer);
            return output.ToString();
        }

        String CreateTemplateForWrite(String fileTemplateText)
        {

            const String tmlHeader =
@"(function() {
    let module = { exports: undefined };
    (function(module, exports) {
";
```

```
            const String tmlFooter =
    @"
    })(module, module.exports);
    return module.exports;
})()";

            var sb = new StringBuilder();

            sb.AppendLine()
            .AppendLine(tmlHeader)
            .AppendLine(fileTemplateText)
            .AppendLine(tmlFooter);
            return sb.ToString();
        }

        HashSet<String> _modulesWritten;

        void AddRequiredModules(StringBuilder sb, String clientScript)
        {
            const String tmlHeader =
@"
    app.modules['$(Module)'] = function() {
    let module = { exports: undefined };
    (function(module, exports) {
";

            const String tmlFooter =
@"
    })(module, module.exports);
    return module.exports;
};";

            if (String.IsNullOrEmpty(clientScript))
                return;
            if (_modulesWritten == null)
                _modulesWritten = new HashSet<String>();
            int iIndex = 0;
            while (true)
            {
                String moduleName = FindModuleNameFromString(clientScript, ref iIndex);
                if (moduleName == null)
                    return; // not found
                if (String.IsNullOrEmpty(moduleName))
                    continue;
                if (moduleName.ToLowerInvariant().StartsWith("global/"))
                    continue;
                if (moduleName.ToLowerInvariant().StartsWith("std:"))
                    continue;
                if (_modulesWritten.Contains(moduleName))
                    continue;
                var fileName = moduleName.AddExtension("js");
                var filePath = Path.GetFullPath(Path.Combine(_host.AppPath, _host.AppKey,
fileName.RemoveHeadSlash()));
                if (!File.Exists(filePath))
                    throw new FileNotFoundException(filePath);
                String moduleText = File.ReadAllText(filePath);
                sb.AppendLine(tmlHeader.Replace("$(Module)", moduleName))
                    .AppendLine(_localizer.Localize(null, moduleText))
                    .AppendLine(tmlFooter)
                    .AppendLine();
                _modulesWritten.Add(moduleName);
                AddRequiredModules(sb, moduleText);
            }
        }
```

```
public static String FindModuleNameFromString(String text, ref int pos)
{
    String funcName = "require";
    int rPos = text.IndexOf(funcName, pos);
    if (rPos == -1)
        return null; // не продолжаем, ничего не нашли
    pos = rPos + funcName.Length;
    // проверим, что мы не в комментарии
    int oc = text.LastIndexOf("/*", rPos);
    int cc = text.LastIndexOf("*/", rPos);
    if (oc != -1)
    {
        // есть открывающий комментарий
        if (cc == -1)
        {
            return String.Empty; // нет закрывающего
        }
        if (cc < oc)
        {
            return String.Empty; // закрывающий левее открывающего, мы внутри
        }
    }
    int startLine = text.LastIndexOfAny(new Char[] { '\r', '\n' }, rPos);
    oc = text.LastIndexOf("//", rPos);
    if ((oc != 1) && (oc > startLine))
        return String.Empty; // есть однострочный и он после начала строки

    Tokenizer tokenizer = null;
    try
    {
        // проверим точку, как предыдущий токен
        var dotPos = text.LastIndexOf('.', rPos);
        if (dotPos != -1)
        {
            tokenizer = new Tokenizer(text, dotPos);
            if (tokenizer.token.id == Tokenizer.TokenId.Dot)
            {
                tokenizer.NextToken();
                var tok = tokenizer.token;
                if (tok.id == Tokenizer.TokenId.Identifier && tok.Text == "require")
                {
                    tokenizer.NextToken();
                    if (tokenizer.token.id == Tokenizer.TokenId.OpenParen)
                        return String.Empty; /* есть точка перед require */
                }
            }
        }
        tokenizer = new Tokenizer(text, rPos + funcName.Length);
        if (tokenizer.token.id == Tokenizer.TokenId.OpenParen)
        {
            tokenizer.NextToken();
            if (tokenizer.token.id == Tokenizer.TokenId.StringLiteral)
            {
                pos = tokenizer.GetTextPos();
                return tokenizer.token.UnquotedText.Replace("\\\\", "/");
            }
        }
        pos = tokenizer.GetTextPos();
        return String.Empty;
    }
    catch (Exception /*ex*/)
    {
        // parser error
```

```
                if (tokenizer != null)
                    pos = tokenizer.GetTextPos();
                return null;
            }
        }

        public static readonly JsonSerializerSettings StandardSerializerSettings =
            new JsonSerializerSettings() {
                Formatting = Formatting.Indented,
                StringEscapeHandling = StringEscapeHandling.EscapeHtml,
                DateFormatHandling = DateFormatHandling.IsoDateFormat,
                DateTimeZoneHandling = DateTimeZoneHandling.Utc,
                NullValueHandling = NullValueHandling.Ignore,
                DefaultValueHandling = DefaultValueHandling.Ignore
            };

        public void ProfileException(Exception ex)
        {
            using (Host.Profiler.CurrentRequest.Start(ProfileAction.Exception, ex.Message))
            {
                // do nothing
            }
        }

        public void WriteHtmlException(Exception ex, TextWriter writer)
        {
            if (ex.InnerException != null)
                ex = ex.InnerException;
            ProfileException(ex);
            var msg = WebUtility.HtmlEncode(ex.Message);
            var stackTrace = WebUtility.HtmlEncode(ex.StackTrace);
            if (IsDebugConfiguration)
                writer.Write($"<div class=\"app-exception\"><div
class=\"message\">{msg}</div><div class=\"stack-trace\">{stackTrace}</div></div>");
            else
                writer.Write($"<div class=\"app-exception\"><div
class=\"message\">{msg}</div></div>");
        }
    }
}
```

Файл bindcmd.cs

```
// Copyright © 2015-2017 Alex Kukhtin. All rights reserved.

using A2v10.Infrastructure;
using System;
using System.Text;

/*
 * $exec(cmd, arg, confirm, opts) : $canExecute(cmd, arg, opts)
 * $dialog(cmd, url, arg, data(query), opts)
 */

namespace A2v10.Xaml
{
    public enum CommandType
    {
        Unknown,
        Close,
        SaveAndClose,
        Reload,
        Refresh,
        Requery,
        Save,
        Create,
        Clear,
        Open,
        OpenSelected,
        DbRemoveSelected,
        DbRemove,
        Append,
        Browse,
        Execute,
        ExecuteSelected,
        Remove,
        RemoveSelected,
        Dialog,
        Select,
        SelectChecked,
        Report,
    }

    public enum DialogAction
    {
        Unknown,
        Edit,
        EditSelected,
        Show,
        Browse,
        Append, // create in dialog and append to array
    }


    public class BindCmd : BindBase
    {
        private const String nullString = "null";

        public CommandType Command { get; set; }
        public String Argument { get; set; }
        public String Url { get; set; }
        public DialogAction Action { get; set; }

        public String Execute { get; set; }
        public String CommandName { get; set; }
```

```
        public String Report { get; set; }

        public Boolean SaveRequired { get; set; }
        public Boolean ValidRequired { get; set; }
        public Boolean CheckReadOnly { get; set; }
        public Boolean NewWindow { get; set; }

        public Confirm Confirm { get; set; }

        public String Data { get; set; }

        public BindCmd()
        {

        }
        public BindCmd(String command)
        {
            if (!Enum.TryParse<CommandType>(command, out CommandType cmdType))
                throw new XamlException($"Invalid command '{command}'");
            Command = cmdType;
        }

        internal String GetHrefForCommand(RenderContext context)
        {
            switch (Command)
            {
                case CommandType.Open:
                    return $"$href({CommandUrl(context)}, {CommandArgument(context)})";
            }
            return null;
        }
        internal String GetCommand(RenderContext context, Boolean indirect = false)
        {
            switch (Command)
            {
                case CommandType.Unknown:
                    throw new NotImplementedException($"Command required for BindCmd
extension");
                case CommandType.Refresh:
                case CommandType.Reload:
                    return $"$reload({CommandArgument(context, nullable:true)})";

                case CommandType.Requery:
                    return "$requery()";

                case CommandType.Save:
                    return "$save()";

                case CommandType.Clear:
                    return $"{CommandArgument(context)}.$empty()";

                case CommandType.Close:
                    return context.IsDialog ? "$modalClose()" : "$close()";

                case CommandType.SaveAndClose:
                    if (context.IsDialog)
                        return $"$modalSaveAndClose(null, {GetOptionsValid(context)})";
                    return "$saveAndClose()";

                case CommandType.OpenSelected:
                    return $"$openSelected({CommandUrl(context, decorate:true)},
{CommandArgument(context)})";
```

```
                case CommandType.Select:
                    return $"$modalSelect({CommandArgument(context)})";

                case CommandType.SelectChecked:
                    return $"$modalSelectChecked({CommandArgument(context)})";

                case CommandType.RemoveSelected:
                    return $"$removeSelected({CommandArgument(context)},
{GetConfirm(context)})";

                case CommandType.DbRemove:
                    return $"$dbRemove({CommandArgument(context)}, {GetConfirm(context)})";

                case CommandType.DbRemoveSelected:
                    return $"$dbRemoveSelected({CommandArgument(context)},
{GetConfirm(context)})";

                case CommandType.Open:
                    if (indirect)
                    {
                        if (!IsArgumentEmpty(context))
                            return $"{{cmd:$navigate, eval: true, arg1:{CommandUrl(context,
true)}, arg2:'{CommandArgument(context)}'}}";
                        return $"{{cmd:$navigate, eval: true, arg1:{CommandUrl(context,
true)}, arg2:'this'}}";
                    }
                    else
                        return $"$navigate({CommandUrl(context)}, {CommandArgument(context)},
{NewWindow.ToString().ToLowerInvariant()})";

                case CommandType.Create:
                    return $"$navigate({CommandUrl(context)})";

                case CommandType.Remove:
                    if (indirect)
                    {
                        return $"{{cmd:$remove, arg1:'this'}}";
                    }
                    else
                        return $"$remove({CommandArgumentOrThis(context)},
{GetConfirm(context)})";

                case CommandType.Append:
                    return $"{CommandArgument(context)}.$append()";

                case CommandType.Browse:
                    return $"$dialog('browse', {CommandUrl(context)},
{CommandArgument(context)}, {GetData(context)})";

                case CommandType.Execute:
                    return $"$exec('{GetName()}', {CommandArgument(context, nullable:true)},
{GetConfirm(context)}, {GetOptions(context)})";

                case CommandType.ExecuteSelected:
                    return $"$execSelected('{GetName()}', {CommandArgument(context)},
{GetConfirm(context)})";

                case CommandType.Report:
                    return $"$report('{GetReportName()}', {CommandArgument(context,
nullable:true)}, {GetOptions(context)})";

                case CommandType.Dialog:
                    if (Action == DialogAction.Unknown)
                        throw new XamlException($"Action required for {Command} command");
```

```
                    String action = Action.ToString().ToKebabCase();
                    Boolean bNullable = false;
                    if (Action == DialogAction.Show)
                        bNullable = true; // Nullable actions ???
                    if (indirect)
                    {
                        String arg3 = "this";
                        if (!IsArgumentEmpty(context))
                            arg3 = CommandArgument(context);
                        // command, url, data
                        return $"{{cmd:$dialog, isDialog:true, arg1:'{action}',
arg2:{CommandUrl(context)}, arg3: '{arg3}'}}";
                    }
                    return $"$dialog('{action}', {CommandUrl(context)},
{CommandArgument(context, bNullable)}, {GetData(context)}, {GetOptions(context)})";

                default:
                    throw new NotImplementedException($"command '{Command}' yet not
implemented");
            }
        }
        String GetName()
        {
            if (String.IsNullOrEmpty(CommandName))
                throw new XamlException($"CommandName required for {Command} command");
            return CommandName;
        }

        String GetReportName()
        {
            if (String.IsNullOrEmpty(Report))
                throw new XamlException($"ReportName required for {Command} command");
            return Report;
        }

        String GetOptions(RenderContext context)
        {
            if (!SaveRequired && !ValidRequired && !CheckReadOnly)
                return nullString;
            StringBuilder sb = new StringBuilder("{");
            if (SaveRequired)
                sb.Append("saveRequired: true,");
            if (ValidRequired)
                sb.Append("validRequired: true,");
            if (CheckReadOnly)
                sb.Append("checkReadOnly: true,");
            sb.RemoveTailComma();
            sb.Append("}");
            return sb.ToString();
        }

        String GetOptionsValid(RenderContext context)
        {
            if (!ValidRequired)
                return String.Empty;
            StringBuilder sb = new StringBuilder("{");
            if (ValidRequired)
            {
                sb.Append("validRequired: true, ");
            }
            sb.RemoveTailComma();
            sb.Append("}");
            return sb.ToString();
        }
```

```
String CommandArgument(RenderContext context, Boolean nullable = false)
{
    String arg = null;
    if (nullable)
    {
        var argBind = GetBinding(nameof(Argument));
        if (argBind != null)
            arg = argBind.GetPath(context);
    }
    else
        arg = ArgumentBinding.GetPath(context);
    if (String.IsNullOrEmpty(arg))
        return nullString;
    return arg;
}

String GetData(RenderContext context)
{
    var dataBind = GetBinding(nameof(Data));
    if (dataBind != null)
        return dataBind.GetPath(context);
    else if (Data != null)
        return Data;
    return nullString;
}

String GetConfirm(RenderContext context)
{
    if (Confirm == null)
        return nullString;
    return Confirm.GetJsValue(context);
}

Boolean IsArgumentEmpty(RenderContext context)
{
    var argBind = GetBinding(nameof(Argument));
    return argBind == null || String.IsNullOrEmpty(argBind.Path);

}

String CommandArgumentOrThis(RenderContext context)
{
    var argBind = GetBinding(nameof(Argument));
    if (argBind != null)
        return argBind.GetPath(context);
    var path = context.GetNormalizedPath(String.Empty);
    if (String.IsNullOrEmpty(path))
        throw new XamlException($"Invalid arguments for {Command} command");
    return path;
}

Bind ArgumentBinding
{
    get
    {
        var arg = GetBinding(nameof(Argument));
        if (arg != null)
            return arg;
        throw new XamlException($"Argument bind required for {Command} command");
    }
}

String CommandUrl(RenderContext context, Boolean decorate = false)
```

```
        {
            var urlBind = GetBinding(nameof(Url));
            if (urlBind != null)
            {
                if (decorate)
                    return $"'{{{urlBind.Path}}}'";
                return urlBind.GetPath(context);
            }
            if (String.IsNullOrEmpty(Url))
                throw new NotImplementedException($"Url required for {Command} command");
            // TODO: check URL format
            if (!Url.StartsWith("/"))
                throw new NotImplementedException($"Url '{Url}' must start with '/'");
            return $"'{Url.ToLowerInvariant()}'";
        }

        internal void MergeCommandAttributes(TagBuilder tag, RenderContext context)
        {
            switch (Command)
            {
                case CommandType.Save:
                case CommandType.SaveAndClose:
                    if (context.IsDataModelIsReadOnly)
                        tag.MergeAttribute(":disabled", "true");
                    else
                        tag.MergeAttribute(":disabled", "$isPristine");
                    break;
                case CommandType.Execute:
                    tag.MergeAttribute(":disabled", $"!$canExecute('{CommandName}',
{CommandArgument(context, true)}, {GetOptions(context)})");
                    break;
                case CommandType.Append:
                case CommandType.Remove:
                    if (context.IsDataModelIsReadOnly)
                        tag.MergeAttribute(":disabled", "true");
                    break;
                case CommandType.SelectChecked:
                    {
                        var arg = GetBinding(nameof(Argument));
                        if (arg != null)
                            tag.MergeAttribute(":disabled",
$"!$hasChecked({arg.GetPath(context)})");
                    }
                    break;
                case CommandType.OpenSelected:
                case CommandType.Select:
                case CommandType.ExecuteSelected:
                case CommandType.DbRemoveSelected:
                    {
                        var arg = GetBinding(nameof(Argument));
                        if (arg != null)
                            tag.MergeAttribute(":disabled",
$"!$hasSelected({arg.GetPath(context)})");
                    }
                    break;
                case CommandType.RemoveSelected:
                    if (context.IsDataModelIsReadOnly)
                        tag.MergeAttribute(":disabled", "true");
                    else
                    {
                        var arg = GetBinding(nameof(Argument));
                        if (arg != null)
                            tag.MergeAttribute(":disabled",
$"!$hasSelected({arg.GetPath(context)})");
```

```
                    }
                    break;
                case CommandType.Dialog:
                    if (Action == DialogAction.EditSelected)
                    {
                        var arg = GetBinding(nameof(Argument));
                        if (arg != null)
                            tag.MergeAttribute(":disabled",
$"!$hasSelected({arg.GetPath(context)})");
                    }
                    break;
            }
        }
    }
}
```

Файл DataGrid.cs

```
// Copyright © 2015-2017 Alex Kukhtin. All rights reserved.

using System;
using System.Windows.Markup;
using A2v10.Infrastructure;

namespace A2v10.Xaml
{
    public enum HeadersVisibility
    {
        Column,
        None
    }

    public enum RowDetailsActivate
    {
        ActiveRow,
        Cell
    }

    [ContentProperty("Columns")]
    public class DataGrid : Control
    {
        public Boolean Hover { get; set; }
        public Boolean Striped { get; set; }
        public Boolean Border { get; set; }
        public Boolean Sort { get; set; }
        public Boolean Compact { get; set; }

        public Boolean FixedHeader { get; set; }
        public HeadersVisibility HeadersVisibility { get; set; }

        public GridLinesVisibility GridLines { get; set; }

        public Object ItemsSource { get; set; }

        public DataGridColumnCollection Columns { get; set; } = new
DataGridColumnCollection();

        public RowMarkerStyle MarkerStyle { get; set; }

        public MarkStyle Mark { get; set; }

        public Boolean? RowBold { get; set; }

        public Command DoubleClick { get; set; }

        public Length Height { get; set; }

        public DataGridRowDetails RowDetails { get; set; }

        GroupDescriptions _groupBy;
        public GroupDescriptions GroupBy
        {
            get
            {
                if (_groupBy == null)
                    _groupBy = new GroupDescriptions();
                return _groupBy;
            }
            set { _groupBy = value; }
        }
```

```
        internal override void RenderElement(RenderContext context, Action<TagBuilder>
onRender = null)
        {
            var dataGrid = new TagBuilder("data-grid", null, IsInGrid);
            if (onRender != null)
                onRender(dataGrid);
            MergeBindingAttributeBool(dataGrid, context, ":compact", nameof(Compact),
Compact);
            MergeAttributes(dataGrid, context, MergeAttrMode.Margin |
MergeAttrMode.Visibility);
            if (Height != null)
                dataGrid.MergeStyle("height", Height.Value);
            if (FixedHeader)
                dataGrid.MergeAttribute(":fixed-header", "true");
            if (HeadersVisibility == HeadersVisibility.None)
                dataGrid.MergeAttribute(":hide-header", "true");
            if (RowDetails != null)
            {
                dataGrid.MergeAttribute(":row-details", "true");
                dataGrid.MergeAttribute("row-details-activate",
RowDetails.Activate.ToString().ToLowerInvariant());
                var vBind = RowDetails.GetBinding("Visible");
                if (vBind != null)
                {
                    dataGrid.MergeAttribute("row-details-visible", vBind.Path /*!without
context!*/);
                }
            }
            var isb = GetBinding(nameof(ItemsSource));
            if (isb != null)
                dataGrid.MergeAttribute(":items-source", isb.GetPath(context));
            MergeBoolAttribute(dataGrid, context, nameof(Hover), Hover);
            MergeBoolAttribute(dataGrid, context, nameof(Striped), Striped);
            MergeBoolAttribute(dataGrid, context, nameof(Border), Border);
            MergeBoolAttribute(dataGrid, context, nameof(Sort), Sort);
            dataGrid.MergeAttribute(":route-query", "$query"); // always!

            var dblClickBind = GetBindingCommand(nameof(DoubleClick));
            if (dblClickBind != null)
            {
                // Function!
                dataGrid.MergeAttribute(":doubleclick", "() => " +
dblClickBind.GetCommand(context));
            }

            if (MarkerStyle != RowMarkerStyle.None)
                dataGrid.MergeAttribute("mark-style", MarkerStyle.ToString().ToKebabCase());

            var mbind = GetBinding(nameof(Mark));
            if (mbind != null)
            {
                dataGrid.MergeAttribute("mark", mbind.GetPath(context));
            }
            else if (Mark != MarkStyle.Default)
            {
                throw new XamlException("The Mark property must be a binding");
            }
            var rbbind = GetBinding(nameof(RowBold));

            if (rbbind != null)
                dataGrid.MergeAttribute("row-bold", rbbind.GetPath(context));
            else if (RowBold != null)
                throw new XamlException("The RowBold property must be a binding");
```

```
            // TODO: binding for GridLines ???
            if (GridLines != GridLinesVisibility.None)
                dataGrid.MergeAttribute("grid", GridLines.ToString());

            var groupByBind = GetBinding(nameof(GroupBy));
            if (groupByBind != null)
            {
                dataGrid.MergeAttribute(":group-by", groupByBind.GetPath(context));
            }
            else if (_groupBy != null)
            {
                dataGrid.MergeAttribute(":group-by", _groupBy.GetJsValue());
            }

            dataGrid.RenderStart(context);
            Int32 colIndex = 0;
            foreach (var col in Columns)
            {
                col.RenderColumn(context, colIndex);
                colIndex++;
            }
            RenderRowDetails(context);
            dataGrid.RenderEnd(context);
        }

        void RenderRowDetails(RenderContext context)
        {
            if (RowDetails == null)
                return;
            var rdtag = new TagBuilder("template");
            rdtag.MergeAttribute("slot", "row-details");
            rdtag.MergeAttribute("slot-scope", "details");
            rdtag.RenderStart(context);
            using (var ctx = new ScopeContext(context, "details.row"))
            {
                RowDetails.Content.RenderElement(context);
            }
            rdtag.RenderEnd(context);

        }

        protected override void OnEndInit()
        {
            base.OnEndInit();
            foreach (var col in Columns)
                col.SetParent(this);
        }
    }
}
```

Файл Table.cs

```
// Copyright © 2015-2017 Alex Kukhtin. All rights reserved.

using System;
using System.Windows.Markup;
using A2v10.Infrastructure;

namespace A2v10.Xaml
{

    public enum TableBackgroundStyle
    {
        None,
        Paper,
        Yellow,
        Cyan,
        Rose
    }

    [ContentProperty("Rows")]
    public class Table : Control, ITableControl
    {
        public GridLinesVisibility GridLines { get; set; }
        public TableRowCollection Rows { get; set; } = new TableRowCollection();
        public Boolean Border { get; set; }
        public Boolean Compact { get; set; }
        public Boolean Hover { get; set; }
        public Boolean Striped { get; set; }

        public TableBackgroundStyle Background { get; set; }

        public TableRowCollection Header
        {
            get
            {
                if (_header == null)
                    _header = new TableRowCollection();
                return _header;
            }
            set
            {
                _header = value;
            }
        }

        public TableRowCollection Footer
        {
            get
            {
                if (_footer == null)
                    _footer = new TableRowCollection();
                return _footer;
            }
            set { _footer = value; }
        }

        public TableColumnCollection Columns
        {
            get
            {
                if (_columns == null)
                    _columns = new TableColumnCollection();
                return _columns;
```

```
            }
            set
            {
                _columns = value;
            }
        }


        TableRowCollection _header;
        TableRowCollection _footer;
        TableColumnCollection _columns;

        public Object ItemsSource { get; set; }

        internal override void RenderElement(RenderContext context, Action<TagBuilder>
onRender = null)
        {
            var table = new TagBuilder("table", "a2-table", IsInGrid);
            if (onRender != null)
                onRender(table);
            MergeAttributes(table, context);
            if (Background != TableBackgroundStyle.None)
                table.AddCssClass("bk-" + Background.ToString().ToKebabCase());

            if (GridLines != GridLinesVisibility.None)
                table.AddCssClass($"grid-{GridLines.ToString().ToLowerInvariant()}");

            table.AddCssClassBool(Border, "bordered");
            table.AddCssClassBool(Compact, "compact");
            table.AddCssClassBool(Hover, "hover");
            table.AddCssClassBool(Striped, "striped");

            Bind isBind = GetBinding(nameof(ItemsSource));
            if (isBind != null)
                table.MergeAttribute("v-lazy", isBind.GetPath(context));
            table.RenderStart(context);
            if (_columns != null)
                Columns.Render(context);
            RenderHeader(context);
            RenderBody(context);
            RenderFooter(context);
            table.RenderEnd(context);
        }

        void RenderHeader(RenderContext context)
        {
            if (_header == null)
                return;
            var thead = new TagBuilder("thead").RenderStart(context);
            foreach (var h in Header)
                h.RenderElement(context);
            thead.RenderEnd(context);
        }

        void RenderBody(RenderContext context)
        {
            if (Rows.Count == 0)
                return;
            var tbody = new TagBuilder("tbody").RenderStart(context);
            Bind isBind = GetBinding(nameof(ItemsSource));
            if (isBind != null)
            {
                var repeatAttr = $"(row, rowIndex) in {isBind.GetPath(context)}";
                using (new ScopeContext(context, "row"))
```

```
                {
                    if (Rows.Count == 1)
                    {
                        Rows[0].RenderElement(context, (tag) =>
                        {
                            tag.MergeAttribute("v-for", repeatAttr);
                        });
                    }
                    else
                    {
                        var tml = new TagBuilder("template");
                        tml.MergeAttribute("v-for", repeatAttr);
                        tml.RenderStart(context);
                        using (var cts = new ScopeContext(context, "row"))
                        {
                            var rNo = 0;
                            foreach (var row in Rows)
                            {
                                row.RenderElement(context, (tag) => tag.MergeAttribute(":key",
$"'r{rNo}:' + rowIndex"));
                                rNo += 1;
                            }
                        }
                        tml.RenderEnd(context);
                    }
                }
            }
            else
            {
                foreach (var row in Rows)
                    row.RenderElement(context);
            }
            tbody.RenderEnd(context);
        }

        void RenderFooter(RenderContext context)
        {
            if (_footer == null)
                return;
            var tfoot = new TagBuilder("tfoot").RenderStart(context);
            foreach (var f in Footer)
                f.RenderElement(context);
            tfoot.RenderEnd(context);
        }

        protected override void OnEndInit()
        {
            base.OnEndInit();
            foreach (var c in Rows)
                c.SetParent(this);
            if (_header != null)
                foreach (var h in Header)
                    h.SetParent(this);
            if (_footer != null)
                foreach (var f in Footer)
                    f.SetParent(this);
            if (_columns != null)
                foreach (var c in Columns)
                    c.SetParent(this);
        }
    }
}
```

Файл datamodel.js

```
// Copyright © 2015-2018 Alex Kukhtin. All rights reserved.
 (function () {
       "use strict";
    /* TODO:
    1. changing event
    4. add plain properties
    */
       const META = '_meta_';
       const PARENT = '_parent_';
       const SRC = '_src_';
       const PATH = '_path_';
       const ROOT = '_root_';
    const ERRORS = '_errors_';

    const ERR_STR = '#err#';

    const FLAG_VIEW = 1;
    const FLAG_EDIT = 2;
    const FLAG_DELETE = 4;

       const platform = require('std:platform');
       const validators = require('std:validators');
       const utils = require('std:utils');
    const log = require('std:log');

    let __initialized__ = false;

       function defHidden(obj, prop, value, writable) {
              Object.defineProperty(obj, prop, {
                     writable: writable || false,
                     enumerable: false,
                     configurable: false,
                     value: value
              });
       }

       function defHiddenGet(obj, prop, get) {
              Object.defineProperty(obj, prop, {
                     enumerable: false,
                     configurable: false,
                     get: get
              });
       }
       function defPropertyGet(trg, prop, get) {
              Object.defineProperty(trg, prop, {
                     enumerable: true,
                     configurable: true, /* needed */
                     get: get
              });
       }

    function ensureType(type, val) {
        if (!utils.isDefined(val))
            val = utils.defaultValue(type);
              if (type === Number) {
                     return utils.toNumber(val);
        }
              return val;
       }

       function defSource(trg, source, prop, parent) {
              let propCtor = trg._meta_.props[prop];
```

```
                let pathdot = trg._path_ ? trg._path_ + '.' : '';
                let shadow = trg._src_;
                source = source || {};
                switch (propCtor) {
                        case Number:
                                shadow[prop] = source[prop] || 0;
                                break;
                        case String:
                                shadow[prop] = source[prop] || "";
                                break;
                        case Boolean:
                                shadow[prop] = source[prop] || false;
                                break;
                        case Date:
                    let srcval = source[prop] || null;
                                shadow[prop] = srcval ? new Date(srcval) : utils.date.zero();
                    break;
                case TMarker: // marker for dynamic property
                    let mp = trg._meta_.markerProps[prop];
                    shadow[prop] = mp;
                    break;
                default:
                                shadow[prop] = new propCtor(source[prop] || null, pathdot + prop,
trg);
                                break;
                }
                Object.defineProperty(trg, prop, {
                        enumerable: true,
                        configurable: true, /* needed */
                get() {
                                return this._src_[prop];
                        },
                        set(val) {
                                //TODO: emit and handle changing event
                                val = ensureType(this._meta_.props[prop], val);
                                if (val === this._src_[prop])
                            return;
                        if (this._src_[prop] && this._src_[prop].$set) {
                            // object
                            this._src_[prop].$merge(val, false);
                        } else {
                            this._src_[prop] = val;
                        }
                        this._root_.$setDirty(true);
                        if (this._lockEvents_)
                            return; // events locked
                                if (!this._path_)
                                        return;
                                let eventName = this._path_ + '.' + prop + '.change';
                                this._root_.$emit(eventName, this, val);
                        }
                });
        }

    function TMarker() { }
    function createPrimitiveProperties(elem, ctor) {
        const templ = elem._root_.$template;
        if (!templ) return;
        const props = templ._props_;
        if (!props) return;
        let objname = ctor.name;
        if (objname in props) {
            for (let p in props[objname]) {
                let propInfo = props[objname][p];
```

```
            if (utils.isPrimitiveCtor(propInfo)) {
                log.info(`create scalar property: ${objname}.${p}`);
                elem._meta_.props[p] = propInfo;
            } else if (utils.isObjectExact(propInfo)) {
                if (!propInfo.get) { // plain object
                    log.info(`create object property: ${objname}.${p}`);
                    elem._meta_.props[p] = TMarker;
                    if (!elem._meta_.markerProps)
                        elem._meta_.markerProps = {};
                    elem._meta_.markerProps[p] = propInfo;
                }
            }
        }
    }
}
   function createObjProperties(elem, ctor) {
          let templ = elem._root_.$template;
          if (!templ) return;
          let props = templ._props_;
          if (!props) return;
          let objname = ctor.name;
    if (objname in props) {
                for (let p in props[objname]) {
            let propInfo = props[objname][p];
            if (utils.isPrimitiveCtor(propInfo)) {
                continue;
            }
            else if (utils.isFunction(propInfo)) {
                log.info(`create property: ${objname}.${p}`);
                Object.defineProperty(elem, p, {
                    configurable: false,
                    enumerable: true,
                    get: propInfo
                });
            } else if (utils.isObjectExact(propInfo)) {
                if (propInfo.get) { // has get, maybe set
                    log.info(`create property: ${objname}.${p}`);
                    Object.defineProperty(elem, p, {
                        configurable: false,
                        enumerable: true,
                        get: propInfo.get,
                        set: propInfo.set
                    });
                }
            } else {
                alert('todo: invalid property type');
            }
                 }
          }
   }

   function createObject(elem, source, path, parent) {
          const ctorname = elem.constructor.name;
          let startTime = null;
          if (ctorname === 'TRoot')
                 startTime = performance.now();
          parent = parent || elem;
          defHidden(elem, SRC, {});
          defHidden(elem, PATH, path);
          defHidden(elem, ROOT, parent._root_ || parent);
          defHidden(elem, PARENT, parent);
          defHidden(elem, ERRORS, null, true);
    defHidden(elem, '_lockEvents_', 0, true);
```

```
let hasTemplProps = false;
const templ = elem._root_.$template;
if (templ && !utils.isEmptyObject(templ._props_))
    hasTemplProps = true;

if (hasTemplProps)
    createPrimitiveProperties(elem, elem.constructor);

      for (let propName in elem._meta_.props) {
             defSource(elem, source, propName, parent);
}

if (hasTemplProps)
    createObjProperties(elem, elem.constructor);

if (path && path.endsWith(']'))
    elem.$selected = false;

defPropertyGet(elem, '$valid', function () {
             if (this._root_._needValidate_)
        this._root_._validateAll_();
             if (this._errors_)
                    return false;
             for (var x in this) {
                    if (x[0] === '$' || x[0] === '_')
                           continue;
                    let sx = this[x];
                    if (utils.isObject(sx) && '$valid' in sx) {
                           let sx = this[x];
                           if (!sx.$valid)
                                  return false;
                    }
             }
             return true;
      });
      defPropertyGet(elem, "$invalid", function () {
             return !this.$valid;
});

if (elem._meta_.$group === true) {
    defPropertyGet(elem, "$groupName", function () {
        if (!utils.isDefined(this.$level))
            return ERR_STR;
        // this.constructor.name == objectType;
        const mi = this._root_.__modelInfo.Levels;
        if (mi) {
            const levs = mi[this.constructor.name];
            if (levs && this.$level <= levs.length);
            return this[levs[this.$level - 1]];
        }
        console.error('invalid data for $groupName');
        return ERR_STR;
    });
}

let constructEvent = ctorname + '.construct';
let _lastCaller = null;
      elem._root_.$emit(constructEvent, elem);
      if (elem._root_ === elem) {
             // root element
             elem._root_ctor_ = elem.constructor;
             elem.$dirty = false;
             elem._query_ = {};
             // rowcount implementation
```

```
                for (var m in elem._meta_.props) {
            let rcp = m + '.$RowCount';
                        if (source && rcp in source) {
                                let rcv = source[rcp] || 0;
                                elem[m].$RowCount = rcv;
                        }
                }
                elem._enableValidate_ = true;
        elem._needValidate_ = false;
        elem._modelLoad_ = (caller) => {
            _lastCaller = caller;
            elem._fireLoad_();
            __initialized__ = true;
        };
        elem._fireLoad_ = () => {
            elem.$emit('Model.load', elem, _lastCaller);
            elem._root_.$setDirty(false);
        };
        defHiddenGet(elem, '$readOnly', isReadOnly);
          }
    if (startTime) {
        log.time('create root time:', startTime, false);
    }
          return elem;
}

function isReadOnly() {
    if ('__modelInfo' in this) {
        let mi = this.__modelInfo;
        if (utils.isDefined(mi.ReadOnly))
            return mi.ReadOnly;
    }
    return false;
}

   function createArray(source, path, ctor, arrctor, parent) {
    let arr = new _BaseArray(source ? source.length : 0);
          let dotPath = path + '[]';
          defHidden(arr, '_elem_', ctor);
          defHidden(arr, PATH, path);
          defHidden(arr, PARENT, parent);
          defHidden(arr, ROOT, parent._root_ || parent);
          defPropertyGet(arr, "$valid", function () {
                if (this._errors_)
                        return false;
                for (var x of this) {
                        if (x._errors_)
                                return false;
                }
                return true;
          });
          defPropertyGet(arr, "$invalid", function () {
                return !this.$valid;
          });

          createObjProperties(arr, arrctor);

          let constructEvent = arrctor.name + '.construct';
          arr._root_.$emit(constructEvent, arr);

          if (!source)
                return arr;
          for (let i = 0; i < source.length; i++) {
                arr[i] = new arr._elem_(source[i], dotPath, arr);
```

```
                arr[i].$checked = false;
            }
            return arr;
    }

function _BaseArray(length) {
    let arr = new Array(length || 0);
    addArrayProps(arr);
    return arr;
    }

//_BaseArray.prototype = Array.prototype;

function addArrayProps(arr) {

    defineCommonProps(arr);

    arr.$new = function (src) {
        let newElem = new this._elem_(src || null, this._path_ + '[]', this);
        newElem.$checked = false;
        return newElem;
    };

    defPropertyGet(arr, "$selected", function () {
        for (let x of this.$elements) {
            if (x.$selected) {
                return x;
            }
        }
        return undefined;
    });

    defPropertyGet(arr, "$elements", function () {
        function* elems(arr) {
            for (let i = 0; i < arr.length; i++) {
                let val = arr[i];
                yield val;
                if (val.$items) {
                    yield* elems(val.$items);
                }
            }
        }
        return elems(this);
    });

    defPropertyGet(arr, "Count", function () {
        return this.length;
    });

    defPropertyGet(arr, "$isEmpty", function () {
        return !this.length;
    });

    defPropertyGet(arr, "$checked", function () {
        return this.filter((el) => el.$checked);
    });

    arr.Selected = function (propName) {
        let sel = this.$selected;
        return sel ? sel[propName] : null;
    };
    arr.$loadLazy = function () {
        return new Promise((resolve, reject) => {
            if (this.$loaded) { resolve(self); return; }
```

```
        if (!this.$parent) { resolve(this); return; }
        const meta = this.$parent._meta_;
        if (!meta.$lazy) { resolve(this); return; }
        let propIx = this._path_.lastIndexOf('.');
        let prop = this._path_.substring(propIx + 1);
        if (!meta.$lazy.indexOf(prop) === -1) { resolve(this); return; }
        this.$vm.$loadLazy(this.$parent, prop).then(() => resolve(this));
    });
}

arr.$append = function (src) {
    const that = this;
    function append(src, select) {
        let addingEvent = that._path_ + '[].adding';
        let newElem = that.$new(src);
        // TODO: emit adding and check result
        let er = that._root_.$emit(addingEvent, that/*array*/, newElem/*elem*/);
        if (er === false)
            return; // disabled
        let len = that.push(newElem);
        let ne = that[len - 1]; // maybe newly created reactive element
        if ('$RowCount' in that) that.$RowCount += 1;
        let eventName = that._path_ + '[].add';
        that._root_.$setDirty(true);
        that._root_.$emit(eventName, that /*array*/, ne /*elem*/, len - 1 /*index*/);
        if (select) {
            ne.$select();
            emitSelect(that, ne);
        }
        // set RowNumber
        if ('$rowNo' in newElem._meta_) {
            let rowNoProp = newElem._meta_.$rowNo;
            newElem[rowNoProp] = len; // 1-based
        }
        return ne;
    }
    if (utils.isArray(src)) {
        let ra = [];
        let lastElem = null;
        src.forEach(function (elem) {
            lastElem = append(elem, false);
            ra.push(lastElem);
        });
        if (lastElem) {
            // last added element
            lastElem.$select();
        }
        return ra;
    } else
        return append(src, true);

};

arr.$empty = function () {
    if (this.$root.isReadOnly)
        return;
    this.splice(0, this.length);
    if ('$RowCount' in this) this.$RowCount = 0;
    return this;
};

arr.$clearSelected = function () {
    let sel = this.$selected;
    if (!sel) return; // already null
```

```
        sel.$selected = false;
        emitSelect(this, null);
    };

    arr.$remove = function (item) {
        if (this.$root.isReadOnly)
            return;
        if (!item)
            return;
        let index = this.indexOf(item);
        if (index === -1)
            return;
        this.splice(index, 1);
        if ('$RowCount' in this) this.$RowCount -= 1;
        // EVENT
        let eventName = this._path_ + '[].remove';
        this._root_.$setDirty(true);
        this._root_.$emit(eventName, this /*array*/, item /*elem*/, index);
        if (!this.length) return;
        if (index >= this.length)
            index -= 1;
        if (this.length > index) {
            this[index].$select();
        }
        // renumber rows
        if ('$rowNo' in item._meta_) {
            let rowNoProp = item._meta_.$rowNo;
            for (let i = 0; i < this.length; i++) {
                this[i][rowNoProp] = i + 1; // 1-based
            }
        }
    };

    arr.$copy = function (src) {
        if (this.$root.isReadOnly)
            return;
        this.$empty();
        if (utils.isArray(src)) {
            for (let i = 0; i < src.length; i++) {
                this.push(this.$new(src[i]));
            }
        }
        return this;
    };
}

   function defineCommonProps(obj) {
          defHiddenGet(obj, "$host", function () {
                 return this._root_._host_;
          });

          defHiddenGet(obj, "$root", function () {
                 return this._root_;
          });

          defHiddenGet(obj, "$parent", function () {
                 return this._parent_;
          });

    defHiddenGet(obj, "$vm", function () {
        if (this._root_ && this._root_._host_)
            return this._root_._host_.$viewModel;
        return null;
          });
```

```
	}

	function defineObject(obj, meta, arrayItem) {
		defHidden(obj.prototype, META, meta);

	 obj.prototype.$merge = merge;
	 obj.prototype.$empty = empty;
	 obj.prototype.$set = setElement;

		defineCommonProps(obj.prototype);

		defHiddenGet(obj.prototype, "$isNew", function () {
			return !this.$id;
		});

	 defHiddenGet(obj.prototype, "$isEmpty", function () {
	     return !this.$id;
	 });

		defHiddenGet(obj.prototype, "$id", function () {
			let idName = this._meta_.$id;
			if (!idName) {
				let tpname = this.constructor.name;
				throw new Error(tpname + ' object does not have an Id property');
			}
			return this[idName];
		});

		defHiddenGet(obj.prototype, "$name", function () {
			let nameName = this._meta_.$name;
			if (!nameName) {
				let tpname = this.constructor.name;
				throw new Error(tpname + ' object does not have a Name
property');
			}
			return this[nameName];
		});

		if (arrayItem) {
			defArrayItem(obj);
		}

		if (meta.$hasChildren) {
			defHiddenGet(obj.prototype, "$hasChildren", function () {
				let hcName = this._meta_.$hasChildren;
				if (!hcName) return undefined;
				return this[hcName];
			});
		}
		if (meta.$items) {
			defHiddenGet(obj.prototype, "$items", function () {
				let itmsName = this._meta_.$items;
				if (!itmsName) return undefined;
				return this[itmsName];
			});
		}
	}

    function emitSelect(arr, item) {
        let selectEvent = arr._path_ + '[].select';
        let er = arr._root_.$emit(selectEvent, arr/*array*/, item);
    }

    function defArrayItem(elem) {
```

```
        elem.prototype.$remove = function () {
                let arr = this._parent_;
                arr.$remove(this);
    };
    elem.prototype.$select = function (root) {
        let arr = root || this._parent_;
        let sel = arr.$selected;
        if (sel === this) return;
        if (sel) sel.$selected = false;
        this.$selected = true;
        emitSelect(arr, this);
          };
   }

   function emit(event, ...arr) {
          if (this._enableValidate_) {
                if (!this._needValidate_) {
                      this._needValidate_ = true;
                }
          }
          log.info('emit: ' + event);
          let templ = this.$template;
          if (!templ) return;
          let events = templ.events;
          if (!events) return;
          if (event in events) {
                // fire event
                log.info('handle: ' + event);
                let func = events[event];
                let rv = func.call(undefined, ...arr);
                if (rv === false)
                      log.info(event + ' returns false');
                return rv;
          }
   }

   function getDelegate(name) {
          let tml = this.$template;
          if (!tml.delegates) {
                console.error('There are no delegates in the template');
                return null;
          }
          if (name in tml.delegates) {
                return tml.delegates[name];
          }
          console.error(`Delegate "${name}" not found in the template`);
   }

function canExecuteCommand(cmd, arg, opts) {
    const tml = this.$template;
    if (!tml) return false;
    if (!tml.commands) return false;
    const cmdf = tml.commands[cmd];
    if (!cmdf) return false;

    const optsCheckValid = opts && opts.validRequired === true;
    const optsCheckRO = opts && opts.checkReadOnly === true;

    if (cmdf.checkReadOnly === true || optsCheckRO) {
        if (this.$root.$readOnly)
            return false;
    }
    if (cmdf.validRequired === true || optsCheckValid) {
```

```
            if (!this.$root.$valid)
                return false;
        }
        if (utils.isFunction(cmdf.canExec)) {
            return cmdf.canExec.call(this, arg);
        } else if (utils.isBoolean(cmdf.canExec)) {
            return cmdf.canExec; // for debugging purposes
        } else if (utils.isDefined(cmdf.canExec)) {
            console.error(`${cmd}.canExec should be a function`);
            return false;
        }
        return true;
    }

    function executeCommand(cmd, arg, confirm, opts) {
           try {
            this._root_._enableValidate_ = false;
            let vm = this.$vm;
            const tml = this.$template;
            if (!tml) return;
            if (!tml.commands) return;
            let cmdf = tml.commands[cmd];
            if (!cmdf) {
                console.error(`Command "${cmd}" not found`);
                return;
            }
            const optConfirm = cmdf.confirm || confirm;
            const optSaveRequired = cmdf.saveRequired || (opts && opts.saveRequired);
            const optValidRequired = cmdf.validRequired || (opts && opts.validRequired);

            if (optValidRequired && !vm.$data.$valid) return; // not valid

            if (utils.isFunction(cmdf.canExec)) {
                if (!cmdf.canExec.call(this, arg)) return;
            }

            let that = this;
            const doExec = function () {
                const realExec = function () {
                    if (utils.isFunction(cmdf))
                        cmdf.call(that, arg);
                    else if (utils.isFunction(cmdf.exec))
                        cmdf.exec.call(that, arg);
                    else
                        console.error($`There is no method 'exec' in command '${cmd}'`);
                }
                if (optConfirm) {
                    vm.$confirm(optConfirm).then(realExec);
                } else {
                    realExec();
                }
            }

            if (optSaveRequired && vm.$isDirty)
                vm.$save().then(doExec);
            else
                doExec();


          } finally {
                this._root_._enableValidate_ = true;
            this._root_._needValidate_ = true;
          }
    }
```

```
    function validateImpl(item, path, val, du) {
              if (!item) return null;
              let tml = item._root_.$template;
              if (!tml) return null;
              var vals = tml.validators;
              if (!vals) return null;
              var elemvals = vals[path];
              if (!elemvals) return null;
              return validators.validate(elemvals, item, val, du);
       }

    function saveErrors(item, path, errors) {
              if (!item._errors_ && !errors)
                     return; // already null
              else if (!item._errors_ && errors)
            item._errors_ = {}; // new empty object
        if (errors && errors.length > 0)
                     item._errors_[path] = errors;
              else if (path in item._errors_)
            delete item._errors_[path];
              if (utils.isEmptyObject(item._errors_))
                     item._errors_ = null;
              return errors;
       }

       function validate(item, path, val, ff) {
              if (!item._root_._needValidate_) {
                     // already done
                     if (!item._errors_)
                            return null;
                     if (path in item._errors_)
                            return item._errors_[path];
                     return null;
        }
              let res = validateImpl(item, path, val, ff);
              return saveErrors(item, path, res);
       }

    function* enumData(root, path, name, index) {
        index = index || '';
              if (!path) {
                     // scalar value in root
                     yield { item: root, val: root[name], ix: index };
                     return;
              }
              let sp = path.split('.');
              let currentData = root;
              for (let i = 0; i < sp.length; i++) {
                     let last = i === sp.length - 1;
                     let prop = sp[i];
                     if (prop.endsWith('[]')) {
                            // is array
                            let pname = prop.substring(0, prop.length - 2);
                            if (!(pname in currentData)) {
                                   console.error(`Invalid validator key. Property '${pname}'
not found in '${currentData.constructor.name}'`);
                            }
                            let objto = currentData[pname];
                            if (!objto) continue;
                            for (let j = 0; j < objto.length; j++) {
                                   let arrItem = objto[j];
                                   if (last) {
```

```
                                        yield { item: arrItem, val: arrItem[name], ix:
index + ':' + j };
                                } else {
                                        let newpath = sp.slice(i + 1).join('.');
                                        yield* enumData(arrItem, newpath, name, index + ':'
+ j);
                                }
                        }
                        return;
                } else {
            // simple element
            if (!(prop in currentData)) {
                console.error(`Invalid Validator key. property '${prop}' not found in
'${currentData.constructor.name}'`);
            }
            let objto = currentData[prop];
                        if (last) {
                                if (objto)
                                        yield { item: objto, val: objto[name], ix: index };
                                return;
                        }
                        else {
                                currentData = objto;
                        }
                }
        }
    }

    // enumerate all data (recursive)
    function* dataForVal(root, path) {
        let ld = path.lastIndexOf('.');
        let dp = '';
        let dn = path;
        if (ld !== -1) {
                dp = path.substring(0, ld);
                dn = path.substring(ld + 1);
        }
        yield* enumData(root, dp, dn, '');
    }

    function validateOneElement(root, path, vals) {
        if (!vals)
                return;
        let errs = [];
        for (let elem of dataForVal(root, path)) {
                let res = validators.validate(vals, elem.item, elem.val);
                saveErrors(elem.item, path, res);
                if (res && res.length) {
                        errs.push(...res);
                        // elem.ix - array indexes
                        // console.dir(elem.ix);
                }

        }
        return errs.length ? errs : null;
    }

    function validateAll() {
        var me = this;
     if (!me._host_) return;
     if (!me._needValidate_) return;
     me._needValidate_ = false;
        var startTime = performance.now();
        let tml = me.$template;
```

```
          if (!tml) return;
          let vals = tml.validators;
          if (!vals) return;
          let allerrs = [];
          for (var val in vals) {
                 let err1 = validateOneElement(me, val, vals[val]);
                 if (err1) {
                        allerrs.push({ x: val, e: err1 });
                 }
          }
          var e = performance.now();
          log.time('validation time:', startTime);
          //console.dir(allerrs);
   }

function setDirty(val) {
    if (this.$root.$readOnly)
        return;
          this.$dirty = val;
   }

function empty() {
    this.$set({});
}

function setElement(src) {
    if (this.$root.isReadOnly)
        return;
    this.$merge(src, true);
}

function merge(src, fireChange) {
    try {
        if (src === null)
            src = {};
        this._root_._enableValidate_ = false;
        this._lockEvents_ += 1;
                 for (var prop in this._meta_.props) {
                        let ctor = this._meta_.props[prop];
                        let trg = this[prop];
                        if (Array.isArray(trg)) {
                               trg.$copy(src[prop]);
                               // copy rowCount
                 if ('$RowCount' in trg) {
                                      let rcProp = prop + '.$RowCount';
                                      if (rcProp in src)
                                             trg.$RowCount = src[rcProp] || 0;
                                      else
                                             trg.$RowCount = 0;
                               }
                               //TODO: try to select old value
            } else {
                if (utils.isDateCtor(ctor))
                    platform.set(this, prop, new Date(src[prop]));
                else if (utils.isPrimitiveCtor(ctor)) {
                    platform.set(this, prop, src[prop]);
                } else {
                                      let newsrc = new ctor(src[prop], prop, this);
                                      platform.set(this, prop, newsrc);
                               }
                        }
                 }
          } finally {
                 this._root_._enableValidate_ = true;
```

```
                this._root_._needValidate_ = true;
            this._lockEvents_ -= 1;
        }
        if (fireChange) {
            // emit .change event for all object
            let eventName = this._path_ + '.change';
            this._root_.$emit(eventName, this.$parent, this);
        }
    }

    function implementRoot(root, template, ctors) {
            root.prototype.$emit = emit;
            root.prototype.$setDirty = setDirty;
            root.prototype.$merge = merge;
            root.prototype.$template = template;
        root.prototype._exec_ = executeCommand;
        root.prototype._canExec_ = canExecuteCommand;
            root.prototype._delegate_ = getDelegate;
            root.prototype._validate_ = validate;
            root.prototype._validateAll_ = validateAll;
            // props cache for t.construct
            if (!template) return;
            let xProp = {};
            for (let p in template.properties) {
                    let px = p.split('.'); // Type.Prop
                    if (px.length !== 2) {
                            console.error(`invalid propery name '${p}'`);
                            continue;
                    }
                    let typeName = px[0];
                    let propName = px[1];
                    let pv = template.properties[p]; // property value
                    if (!(typeName in xProp))
                            xProp[typeName] = {};
                    xProp[typeName][propName] = pv;
            }
        template._props_ = xProp;
        /*
        platform.defer(() => {
            console.dir('end init');
        });
        */
    }

    function setModelInfo(root, info) {
            // may be default
            root.__modelInfo = info ? info : {
                    PageSize: 20
            };
    }

    app.modules['std:datamodel'] = {
            createObject: createObject,
            createArray: createArray,
            defineObject: defineObject,
            implementRoot: implementRoot,
            setModelInfo: setModelInfo,
            enumData: enumData
    };
})();
```

Файл datagrid.js

```
// Copyright © 2015-2018 Alex Kukhtin. All rights reserved.

// 20180218-7118
// components/datagrid.js*/

(function () {

 /*TODO:
7. Доделать checked
10.
*/

/*some ideas from
https://github.com/andrewcourtice/vuetiful/tree/master/src/components/datatable */

       /**
        * группировки. v-show на строке гораздо быстрее, чем v-if на всем шаблоне
        */

       /*
              {{g.group}} level:{{g.level}} expanded:{{g.expanded}} source:{{g.source}} count:
        */


       const utils = require('std:utils');
       const log = require('std:log');

    const dataGridTemplate = `
<div v-lazy="itemsSource" :class="{'data-grid-container':true, 'fixed-header': fixedHeader,
'bordered': border}">
    <div :class="{'data-grid-body': true, 'fixed-header': fixedHeader}">
    <table :class="cssClass">
        <colgroup>
            <col v-if="isMarkCell" class="fit"/>
                     <col v-if="isGrouping" class="fit"/>
            <col v-if="isRowDetailsCell" class="fit" />
            <col v-bind:class="columnClass(col)" v-bind:style="columnStyle(col)" v-for="(col,
colIndex) in columns" :key="colIndex"></col>
        </colgroup>
        <thead>
            <tr v-show="isHeaderVisible">
                <th v-if="isMarkCell" class="marker"><div v-if="fixedHeader" class="h-
holder"> </div></th>
                <th v-if="isRowDetailsCell" class="details-marker"><div v-if="fixedHeader"
class="h-holder"> </div></th>
                            <th v-if="isGrouping" class="group-cell">
                                   <a @click.prevent="expandGroups(gi)" v-for="gi in
$groupCount" v-text='gi' /><a
                                          @click.prevent="expandGroups($groupCount + 1)" v-
text='$groupCount + 1' />
                            </th>
                <slot></slot>
            </tr>
        </thead>
              <template v-if="isGrouping">
                     <tbody>
                            <template v-for="(g, gIndex) of $groups">
                                   <tr v-if="isGroupGroupVisible(g)" :class="'group lev-' +
g.level" :key="gIndex">
                                          <td @click.prevent='toggleGroup(g)'
:colspan="columns.length + 1">
```

```
                                        <span :class="{expmark: true, expanded:
g.expanded}" />
                                        <span class="grtitle" v-text="groupTitle(g)" />
                                        <span v-if="g.source.count" class="grcount" v-
text="g.count" /></td>
                                    </tr>
                                    <template v-for="(row, rowIndex) in g.items">
                                        <data-grid-row v-show="isGroupBodyVisible(g)"
:group="true" :level="g.level" :cols="columns" :row="row" :key="gIndex + ':' + rowIndex"
:index="rowIndex" :mark="mark"></data-grid-row>
                        <data-grid-row-details v-if="rowDetails" :cols="columns.length"
:row="row" :key="'rd:' + gIndex + ':' + rowIndex" :mark="mark">
                            <slot name="row-details" :row="row"></slot>
                        </data-grid-row-details>
                                    </template>
                                </template>
                        </tbody>
                </template>
                <template v-else>
                        <tbody>
                  <template v-for="(item, rowIndex) in $items">
                                <data-grid-row :cols="columns" :row="item" :key="rowIndex"
:index="rowIndex" :mark="mark" />
                      <data-grid-row-details v-if="rowDetails" :cols="columns.length"
:row="item" :key="'rd:' + rowIndex" :mark="mark">
                          <slot name="row-details" :row="item"></slot>
                      </data-grid-row-details>
                  </template>
                        </tbody>
                </template>
                <slot name="footer"></slot>
    </table>
    </div>
</div>
`;

       /* @click.prevent disables checkboxes & other controls in cells */
    const dataGridRowTemplate = `
<tr @click="rowSelect(row)" :class="rowClass()" v-on:dblclick.prevent="doDblClick">
    <td v-if="isMarkCell" class="marker">
        <div :class="markClass"></div>
    </td>
    <td v-if="detailsMarker" class="details-marker" @click.prevent="toggleDetails">
        <i v-if="detailsIcon" class="ico" :class="detailsExpandClass" />
    </td>
       <td class="group-marker" v-if="group"></td>
    <data-grid-cell v-for="(col, colIndex) in cols" :key="colIndex" :row="row" :col="col"
:index="index" />
</tr>`;

    const dataGridRowDetailsTemplate = `
<tr v-if="visible()" class="row-details">
    <td v-if="isMarkCell" class="marker">
        <div :class="markClass"></div>
    </td>
    <td :colspan='totalCols' class="details-cell">
        <div class="details-wrapper"><slot></slot></div>
    </td>
</tr>
`;
    /**
        icon on header!!!
              <i :class="\'ico ico-\' + icon" v-if="icon"></i>
     */
```

```
    const dataGridColumnTemplate = `
<th :class="cssClass" @click.prevent="doSort">
    <div class="h-fill" v-if="fixedHeader">
        {{headerText}}
    </div><div class="h-holder">
              <slot>{{headerText}}</slot>
       </div>
</th>
`;

    const dataGridColumn = {
        name: 'data-grid-column',
        template: dataGridColumnTemplate,
        props: {
            header: String,
                      content: String,
            dataType: String,
            hideZeros: Boolean,
            icon: String,
            bindIcon: String,
            id: String,
            align: { type: String, default: 'left' },
            editable: { type: Boolean, default: false },
            noPadding: { type: Boolean, default: false },
            validate: String,
            sort: { type: Boolean, default: undefined },
                      mark: String,
                      controlType: String,
                      width: String,
            fit: Boolean,
            wrap: String,
            command: Object,
        },
        created() {
                      this.$parent.$addColumn(this);
        },
              computed: {
            dir() {
                             return this.$parent.sortDir(this.content);
            },
            fixedHeader() {
                this.$parent.fixedHeader;
            },
            isSortable() {
                if (!this.content)
                    return false;
                return typeof this.sort === 'undefined' ? this.$parent.isGridSortable :
this.sort;
            },
            isUpdateUrl() {
                return !this.$root.inDialog;
            },
                      template() {
                             return this.id ? this.$parent.$scopedSlots[this.id] : null;
                      },
                      classAlign() {
                             return this.align !== 'left' ? (' text-' +
this.align).toLowerCase() : '';
                      },
            cssClass() {
                let cssClass = this.classAlign;
                if (this.isSortable) {
                    cssClass += ' sort';
                    if (this.dir)
```

```
                cssClass += ' ' + this.dir;
            }
            return cssClass;
        },
        headerText() {
            return this.header || '\xa0';
        }
    },
      methods: {
        doSort() {
            if (!this.isSortable)
                        return;
                this.$parent.doSort(this.content);
        },
        cellCssClass(row, editable) {
            let cssClass = this.classAlign;
            if (this.mark) {
                let mark = row[this.mark];
                if (mark)
                    cssClass += ' ' + mark;
            }
            if (editable && this.controlType !== 'checkbox')
                cssClass += ' cell-editable';
            if (this.wrap)
                cssClass += ' ' + this.wrap;
            return cssClass.trim();
        }
    }
   };

Vue.component('data-grid-column', dataGridColumn);

const dataGridCell = {
    functional: true,
    name: 'data-grid-cell',
    props: {
        row: Object,
        col: Object,
        index: Number
    },
    render(h, ctx) {
        //console.warn('render cell');
        let tag = 'td';
        let row = ctx.props.row;
        let col = ctx.props.col;
                let ix = ctx.props.index;
        let cellProps = {
            'class': col.cellCssClass(row, col.editable || col.noPadding)
        };

        let childProps = {
            props: {
                row: row,
                col: col
            }
        };
        if (col.template) {
            let vNode = col.template(childProps.props);
            return h(tag, cellProps, [vNode]);
        }

                if (col.controlType === 'validator') {
                        let cellValid = {
                                props: ['item', 'col'],
```

```
                                    template: '<span><i v-if="item.$invalid" class="ico ico-
error"></i></span>'
                            };
                            cellProps.class = { 'cell-validator': true };
                            return h(tag, cellProps, [h(cellValid, { props: { item: row, col:
col } })]);
                        }

            if (!col.content && !col.icon && !col.bindIcon) {
                return h(tag, cellProps);
            }

            let validator = {
                props: ['path', 'item'],
                template: '<validator :path="path" :item="item"></validator>'
            };

            let validatorProps = {
                props: {
                    path: col.validate,
                    item: row
                }
            };

                         function normalizeArg(arg, eval) {
                                arg = arg || '';
                    if (arg === 'this')
                        arg = row;
                    else if (arg.startsWith('{')) {
                        arg = arg.substring(1, arg.length - 1);
                        if (!(arg in row))
                            throw new Error(`Property '${arg1}' not found in
${row.constructor.name} object`);
                        arg = row[arg];
                    } else if (arg && eval) {
                        console.error(col.hideZeros);
                        arg = utils.eval(row, arg, col.dataType, col.hideZeros);
                    }
                                return arg;
                         }

                         if (col.command) {
                                // column command -> hyperlink
                                // arg1. command
                                let arg1 = normalizeArg(col.command.arg1, false);
                                let arg2 = normalizeArg(col.command.arg2, col.command.eval);
                    let arg3 = normalizeArg(col.command.arg3, false);
                    let ev = col.command.$ev;
                                let child = {
                                        props: ['row', 'col'],
                                        /*@click.prevent, no stop*/
                        template: '<a @click.prevent="doCommand($event)" :href="getHref()"><i v-
if="hasIcon" :class="iconClass" class="ico"></i><span v-text="eval(row, col.content,
col.dataType, col.hideZeros)"></span></a>',
                        computed: {
                            hasIcon() { return col.icon || col.bindIcon; },
                            iconClass() {
                                if (col.bindIcon)
                                    return 'ico-' + utils.eval(row, col.bindIcon);
                                else if (col.icon)
                                    return 'ico-' + col.icon;
                                return null;
                            }
                        },
```

```
                                methods: {
                        doCommand(ev) {
                            if (ev) {
                                // ??? lock double click ???
                                //ev.stopImmediatePropagation();
                                //ev.preventDefault();
                            }
                                                    col.command.cmd(arg1, arg2, arg3);
                                                },
                                                eval: utils.eval,
                        getHref() {
                            if (col.command && col.command.isDialog)
                                                    return null;
                                                let id = arg2;
                                                if (utils.isObjectExact(arg2))
                                                    id = arg2.$id;
                                                return arg1 + '/' + id;
                                                }
                                            }
                                 };
                                 return h(tag, cellProps, [h(child, childProps)]);
                        }
                /* simple content */
                if (col.content === '$index')
                    return h(tag, cellProps, [ix + 1]);

                function isNegativeRed(col) {
                    if (col.dataType === 'Number' || col.dataType === 'Currency')
                        if (utils.eval(row, col.content, col.dataType, col.hideZeros) < 0)
                            return true;
                    return false;
                }

                let content = utils.eval(row, col.content, col.dataType, col.hideZeros);
                let chElems = [h('span', { 'class': { 'negative-red': isNegativeRed(col) } },
content)];
                let icoSingle = !col.content ? ' ico-single' : '';
                if (col.icon)
                    chElems.unshift(h('i', { 'class': 'ico ico-' + col.icon + icoSingle }));
                else if (col.bindIcon)
                    chElems.unshift(h('i', { 'class': 'ico ico-' + utils.eval(row, col.bindIcon) +
icoSingle }));
                /*TODO: validate ???? */
                          if (col.validate) {
                    chElems.push(h(validator, validatorProps));
                }
                return h(tag, cellProps, chElems);
                  }
        };

        const dataGridRow = {
            name: 'data-grid-row',
            template: dataGridRowTemplate,
            components: {
                'data-grid-cell': dataGridCell
            },
            props: {
                row: Object,
                cols: Array,
                index: Number,
                          mark: String,
                          group: Boolean,
                          level : Number
            },
```

```
    computed: {
        isMarkCell() {
            return this.$parent.isMarkCell;
        },
        detailsMarker() {
            return this.$parent.isRowDetailsCell;
        },
        detailsIcon() {
            if (!this.detailsMarker)
                return false;
            let prdv = this.$parent.rowDetailsVisible;
            if (prdv === false) return true; // property not specified
            return prdv && this.row[prdv];
        },
        detailsExpandClass() {
            return this.row.$details ? "ico-minus-circle" : "ico-plus-circle";
        },
        totalColumns() {
            console.error('implement me');
        },
        markClass() {
           return this.mark ? this.row[this.mark] : '';
        }
    },
    methods: {
        rowClass() {
            let cssClass = '';
            const isActive = this.row.$selected; //this.row == this.$parent.selected();
            if (isActive) cssClass += 'active';
            if (this.$parent.isMarkRow && this.mark) {
                cssClass += ' ' + this.row[this.mark];
            }
            if ((this.index + 1) % 2)
                cssClass += ' even'
            if (this.$parent.rowBold && this.row[this.$parent.rowBold])
                cssClass += ' bold';
            if (this.level)
                cssClass += ' lev-' + this.level;
            return cssClass.trim();
        },
        rowSelect(row) {
            row.$select();
        },
        doDblClick($event) {
                        // deselect text
                        $event.stopImmediatePropagation();
                        if (!this.$parent.doubleclick)
                               return;
            window.getSelection().removeAllRanges();
                        this.$parent.doubleclick();
        },
        toggleDetails($event) {
            //$event.stopImmediatePropagation();
            if (!this.detailsIcon) return;
            Vue.set(this.row, "$details", !this.row.$details);
        }
    }
};

const dataGridRowDetails = {
    name: 'data-grid-row-details',
    template: dataGridRowDetailsTemplate,
    props: {
        cols: Number,
```

```
        row: Object,
        mark: String
    },
    computed: {
        isMarkCell() {
            return this.$parent.isMarkCell;
        },
        markClass() {
            return this.mark ? this.row[this.mark] : '';
        },
        detailsMarker() {
            return this.$parent.isRowDetailsCell;
        },
        totalCols() {
            return this.cols +
                (this.isMarkCell ? 1 : 0) +
                (this.detailsMarker ? 1 : 0);
        }
    },
    methods: {
        visible() {
            if (this.$parent.isRowDetailsCell)
                return this.row.$details ? true : false;
            return this.row == this.$parent.selected();
        }
    }
};

   Vue.component('data-grid', {
          props: {
                 'items-source': [Object, Array],
                 border: Boolean,
                 grid: String,
                 striped: Boolean,
        fixedHeader: Boolean,
        hideHeader: Boolean,
        hover: { type: Boolean, default: false },
        compact: Boolean,
                 sort: Boolean,
                 routeQuery: Object,
                 mark: String,
                 filterFields: String,
        markStyle: String,
        rowBold: String,
                 doubleclick: Function,
        groupBy: [Array, Object],
        rowDetails: Boolean,
        rowDetailsActivate: String,
        rowDetailsVisible: [String /*path*/, Boolean]
          },
          template: dataGridTemplate,
          components: {
        'data-grid-row': dataGridRow,
        'data-grid-row-details': dataGridRowDetails
          },
          data() {
                 return {
                        columns: [],
                        clientItems: null,
                        clientGroups: null,
                        localSort: {
                               dir: 'asc',
                               order: ''
                        }
```

```
                };
        },
        computed: {
                $items() {
                        return this.clientItems ? this.clientItems : this.itemsSource;
                },
                isMarkCell() {
                        return this.markStyle === 'marker' || this.markStyle === 'both';
        },
        isRowDetailsCell() {
            return this.rowDetails && this.rowDetailsActivate == 'cell';
        },
                isMarkRow() {
                        return this.markStyle === 'row' || this.markStyle === 'both';
        },
        isHeaderVisible() {
            return !this.hideHeader;
        },
                cssClass() {
                        let cssClass = 'data-grid';
                        if (this.grid) cssClass += ' grid-' + this.grid.toLowerCase();
                        if (this.striped) cssClass += ' striped';
            if (this.hover) cssClass += ' hover';
            if (this.compact) cssClass += ' compact';
                        return cssClass;
                },
                isGridSortable() {
                        return !!this.sort;
                },
                isLocal() {
                        return !this.$parent.sortDir;
                },
                isGrouping() {
                        return this.groupBy;
                },
                $groupCount() {
                        if (utils.isObjectExact(this.groupBy))
                                return 1;
                        else
                                return this.groupBy.length;
                },
                $groups() {
                        function* enumGroups(src, p0, lev, cnt) {
                                for (let grKey in src) {
                                        if (grKey === 'items') continue;
                                        let srcElem = src[grKey];
                                        let count = srcElem.items ? srcElem.items.length :
0;
                                        if (cnt)
                                                cnt.c += count;
                                        let pElem = {
                                                group: grKey,
                                                p0: p0,
                                                expanded: true,
                                                level: lev,
                                                items: srcElem.items || null,
                                                count: count
                                        };
                                        yield pElem;
                                        if (!src.items) {
                                                let cnt = { c: 0 };
                                                yield* enumGroups(srcElem, pElem, lev + 1,
cnt);
                                                pElem.count += cnt.c;
```

```
                                        }
                                }
                        }
                        //console.dir(this.clientGroups);
                                this.doSortLocally();
                                // classic tree
                                let startTime = performance.now();
                                let grmap = {};
                                let grBy = this.groupBy;
                                if (utils.isObjectExact(grBy))
                                        grBy = [grBy];
                                for (let itm of this.$items) {
                                        let root = grmap;
                                        for (let gr of grBy) {
                                                let key = itm[gr.prop];
                                                if (!utils.isDefined(key)) key = '';
                                                if (key === '') key = "Unknown";
                                                if (!(key in root)) root[key] = {};
                                                root = root[key];
                                        }
                                        if (!root.items)
                                                root.items = [];
                                        root.items.push(itm);
                                }
                                // tree to plain array
                                let grArray = [];
                                for (let el of enumGroups(grmap, null, 1)) {
                                        el.source = grBy[el.level - 1];
                                        if (el.source.expanded === false)
                                                el.expanded = false;
                                        grArray.push(el);
                                }
                                this.clientGroups = grArray;
                                log.time('datagrid grouping time:', startTime);
                                return this.clientGroups;
                        }
                },
                watch: {
                        localSort: {
                                handler() {
                                        this.handleSort();
                                },
                                deep: true
                        },
                        'itemsSource.length'() {
                                this.handleSort();
                        }
                },
        methods: {
            selected() {
                let src = this.itemsSource;
                if (src.$origin) {
                    src = src.$origin;
                }
                return src.$selected;
            },
            $addColumn(column) {
                this.columns.push(column);
            },
                        columnClass(column) {
                                let cls = '';
                                if (column.fit || (column.controlType === 'validator'))
                                        cls += 'fit';
                                if (utils.isDefined(column.dir))
```

```
                                cls += ' sorted';
            return cls;
        },
        columnStyle(column) {
            return {
                width: utils.isDefined(column.width) ? column.width : undefined
            };
                },
                doSort(order) {
                        // TODO: // collectionView || locally
                        if (this.isLocal) {
                                if (this.localSort.order === order)
                                        this.localSort.dir = this.localSort.dir === 'asc' ?
'desc' : 'asc';
                                else {
                                        this.localSort = { order: order, dir: 'asc' };
                                }
                        } else {
                                this.$parent.$emit('sort', order);
                        }
                },
                sortDir(order) {
                        // TODO:
                        if (this.isLocal)
                                return this.localSort.order === order ? this.localSort.dir
: undefined;
                        else
                                return this.$parent.sortDir(order);
                },
        doSortLocally()
                {
                        if (!this.isLocal) return;
                        if (!this.localSort.order) return;
                        let startTime = performance.now();
                        let rev = this.localSort.dir === 'desc';
                        let sortProp = this.localSort.order;
            let arr = [].concat(this.itemsSource);
            arr.sort((a, b) => {
                let av = a[sortProp];
                let bv = b[sortProp];
                if (av === bv)
                    return 0;
                else if (av < bv)
                    return rev ? 1 : -1;
                else
                    return rev ? -1 : 1;
                        });
                        log.time('datagrid sorting time:', startTime);
            this.clientItems = arr;
                },
                handleSort() {
                        if (this.isGrouping)
                                this.clientGroups = null;
                        else
                                this.doSortLocally();
                },
                toggleGroup(g) {
                        g.expanded = !g.expanded;
                },
                isGroupGroupVisible(g) {
                        if (!g.group)
                                return false;
                        if (!g.p0)
                                return true;
```

```
                        let cg = g.p0;
                        while (cg) {
                                if (!cg.expanded) return false;
                                cg = cg.p0;
                        }
                        return true;
                },
                isGroupBodyVisible(g) {
                        if (!g.expanded) return false;
                        let cg = g.p0;
                        while (cg) {
                                if (!cg.expanded) return false;
                                cg = cg.p0;
                        }
                        return true;
                },
                groupTitle(g) {
                        if (g.source && g.source.title)
                                return g.source.title
                                        .replace('{Value}', g.group)
                                        .replace('{Count}', g.count);
                        return g.group;
                },
                expandGroups(lev) {
                        // lev 1-based
                        for (var gr of this.$groups)
                                gr.expanded = gr.level < lev;
                }
        }
    });
})();
```

Файл textbox.js

```
// Copyright © 2015-2018 Alex Kukhtin. All rights reserved.

/*20180114-7091*/
/*components/textbox.js*/

(function () {

    const utils = require('std:utils');

    let textBoxTemplate =
`<div :class="cssClass()">
       <label v-if="hasLabel" v-text="label" />
       <div class="input-group">
              <input ref="input" :type="controlType" v-focus
            v-bind:value="modelValue" v-on:change="updateValue($event.target.value)"
:class="inputClass" :placeholder="placeholder" :disabled="disabled" :tabindex="tabIndex"/>
              <slot></slot>
              <validator :invalid="invalid" :errors="errors"
:options="validatorOptions"></validator>
       </div>
       <span class="descr" v-if="hasDescr" v-text="description"></span>
</div>
`;

    let textAreaTemplate =
        `<div :class="cssClass()">
       <label v-if="hasLabel" v-text="label" />
       <div class="input-group">
              <textarea v-focus v-auto-size="autoSize" v-model.lazy="item[prop]" :rows="rows"
:class="inputClass" :placeholder="placeholder" :disabled="disabled" :tabindex="tabIndex"/>
              <slot></slot>
              <validator :invalid="invalid" :errors="errors"
:options="validatorOptions"></validator>
       </div>
       <span class="descr" v-if="hasDescr" v-text="description"></span>
</div>
`;

    let staticTemplate =
`<div :class="cssClass()">
       <label v-if="hasLabel" v-text="label" />
       <div class="input-group static">
              <span v-focus v-text="text" :class="inputClass" :tabindex="tabIndex"/>
              <slot></slot>
              <validator :invalid="invalid" :errors="errors"
:options="validatorOptions"></validator>
       </div>
       <span class="descr" v-if="hasDescr" v-text="description"></span>
</div>
`;

       /*
       <span>{{ path }}</span>
              <button @click="test" >*</button >
       */

    let baseControl = component('control');

    Vue.component('textbox', {
        extends: baseControl,
        template: textBoxTemplate,
              props: {
```

```
                    item: {
                           type: Object, default() {
                                  return {};
                           }
                    },
            prop: String,
            itemToValidate: Object,
            propToValidate: String,
            placeholder: String,
            password: Boolean
        },
        computed: {
            controlType() {
                return this.password ? "password" : "text";
            }
        },
        methods: {
            updateValue(value) {
                this.item[this.prop] = utils.parse(value, this.dataType);
                if (this.$refs.input.value != this.modelValue) {
                    this.$refs.input.value = this.modelValue;
                    this.$emit('change', this.item[this.prop]);
                }
            }
        }
    });

    Vue.component('a2-textarea', {
        extends: baseControl,
        template: textAreaTemplate,
        props: {
            item: {
                type: Object, default() {
                    return {};
                }
            },
            prop: String,
            itemToValidate: Object,
            propToValidate: String,
            placeholder: String,
            autoSize: Boolean,
            rows:Number
        }
    });

    Vue.component('static', {
        extends: baseControl,
        template: staticTemplate,
        props: {
            item: {
                type: Object, default() {
                    return {};
                }
            },
            prop: String,
            itemToValidate: Object,
            propToValidate: String,
            text: [String, Number, Date]
        }
    });

})();
```

Файл selector.js

```
// Copyright © 2015-2018 Alex Kukhtin. All rights reserved.

// 20180206-7104
// components/selector.js

/* TODO:
    7. create element text and command
    8. scrollIntoView for template (table)
    9.
*/

(function () {
    const popup = require('std:popup');
    const utils = require('std:utils');
    const platform = require('std:platform');

    const baseControl = component('control');

    const DEFAULT_DELAY = 300;

    Vue.component('a2-selector', {
        extends: baseControl,
        template: `
<div :class="cssClass2()">
       <label v-if="hasLabel" v-text="label" />
    <div class="input-group">
        <input v-focus v-model="query" :class="inputClass" :placeholder="placeholder"
            @input="debouncedUpdate"
            @blur.stop="cancel"
            @keydown.stop="keyUp"
            :disabled="disabled" />
              <slot></slot>
              <validator :invalid="invalid" :errors="errors"
:options="validatorOptions"></validator>
        <div class="selector-pane" v-if="isOpen" ref="pane" :style="paneStyle">
            <slot name='pane' :items="items" :is-item-active="isItemActive" :item-
name="itemName" :hit="hit">
                <ul class="selector-pane" :style="listStyle">
                    <li @mousedown.prevent="hit(itm)" :class="{active:
isItemActive(itmIndex)}"
                        v-for="(itm, itmIndex) in items" :key="itmIndex" v-
text="itemName(itm)">}</li>
                </ul>
                <a class="create-elem a2-hyperlink a2-inline"><i class="ico ico-plus"/> новый
элемент</a>
            </slot>
        </div>
    </div>
       <span class="descr" v-if="hasDescr" v-text="description"></span>
</div>
`,
        props: {
            item: Object,
            prop: String,
            display: String,
            itemToValidate: Object,
            propToValidate: String,
            placeholder: String,
            delay: Number,
            minChars: Number,
            fetch: Function,
            listWidth: String,
```

```
    listHeight: String
},
data() {
    return {
        isOpen: false,
        loading: false,
        items: [],
        query: '',
        filter: '',
        current: -1
    };
},
computed: {
    $displayProp() {
        return this.display;
    },
    valueText() {
        return this.item ? this.item[this.prop][this.$displayProp] : '';
    },
    pane() {
        return {
            items: this.items,
            isItemActive: this.isItemActive,
            itemName: this.itemName,
            hit: this.hit
        };
    },
    paneStyle() {
        if (this.listWidth)
            return { width: this.listWidth, minWidth: this.listWidth };
        return null;
    },
    listStyle() {
        if (this.listHeight)
            return { maxHeight: this.listHeight };
        return null;
    },
    debouncedUpdate() {
        let delay = this.delay || DEFAULT_DELAY;
        return utils.debounce(() => {
            this.current = -1;
            this.filter = this.query;
            this.update();
        }, delay);
    }
},
watch: {
    valueText(newVal) {
        this.query = this.valueText;
    }
},
methods: {
    __clickOutside() {
        this.isOpen = false;
    },
    cssClass2() {
        let cx = this.cssClass();
        if (this.isOpen)
            cx += ' open'
        return cx;
    },
    isItemActive(ix) {
        return ix === this.current;
    },
```

```
itemName(itm) {
    return itm[this.$displayProp];
},
cancel() {
    this.query = this.valueText;
    this.isOpen = false;
},
keyUp(event) {
    if (!this.isOpen) return;
    switch (event.which) {
        case 27: // esc
            this.cancel();
            break;
        case 13: // enter
            if (this.current == -1) return;
            this.hit(this.items[this.current]);
            break;
        case 40: // down
            event.preventDefault();
            this.current += 1;
            if (this.current >= this.items.length)
                this.current = 0;
            this.scrollIntoView();
            break;
        case 38: // up
            event.preventDefault();
            this.current -= 1;
            if (this.current < 0)
                this.current = this.items.length - 1;
            this.scrollIntoView();
            break;
        default:
            return;
    }
},
hit(itm) {
    Vue.set(this.item, this.prop, itm);
    this.query = this.valueText;
    this.isOpen = false;
},
scrollIntoView() {
    this.$nextTick(() => {
        let pane = this.$refs['pane'];
        if (!pane) return;
        let elem = pane.querySelector('.active');
        if (!elem) return;
        let pe = elem.parentElement;
        let t = elem.offsetTop;
        let b = t + elem.offsetHeight;
        let pt = pe.scrollTop;
        let pb = pt + pe.clientHeight;
        if (t < pt)
            pe.scrollTop = t;
        if (b > pb)
            pe.scrollTop = b - pe.clientHeight;
        //console.warn(`t:${t}, b:${b}, pt:${pt}, pb:${pb}`);
    });
},
update() {
    let text = this.query || '';
    let chars = +(this.minChars || 0);
    if (chars && text.length < chars) return;
    this.isOpen = true;
    this.loading = true;
```

```
                this.fetchData(text).then((result) => {
                    this.loading = false;
                    // first property from result
                    let prop = Object.keys(result)[0];
                    this.items = result[prop];
                });
            },
            fetchData(text) {
                let elem = this.item[this.prop];
                return this.fetch.call(elem, elem, text);
            }
        },
        mounted() {
            popup.registerPopup(this.$el);
            this.query = this.valueText;
            this.$el._close = this.__clickOutside;
        },
        beforeDestroy() {
            popup.unregisterPopup(this.$el);
        }
    });
})();
```

Файл datagrid.less

```
// Copyright © 2015-2017 Alex Kukhtin. All rights reserved.

/*
    TODO:
    1. do somthing with border: (td:first-child (left:none???)
*/


@dg-header-bg-color: #f5f5f5;

@dg-sort-column-bg-color: #eee;
@dg-header-hover-color: darken(@dg-header-bg-color, 10%);
@dg-header-txt-color: #444;
@dg-bage-bg-color: darken(@table-hdr-brd-color, 20%);


//TODO: from @cmn-xxxxx-brd-color
@dg-danger-bg-color: rgba(254, 220, 220, 0.5);
@dg-danger-brd-color: #fc7c7c;
@dg-warning-bg-color: rgba(255, 216, 0, 0.15);
@dg-warning-brd-color: gold;
@dg-success-bg-color: rgba(216, 255, 216, 0.5);
@dg-success-brd-color: #89e489;
@dg-info-bg-color: rgba(114, 234, 234, 0.15);
@dg-info-brd-color: #72eaea;

@group-padding: 12px;
@group-bg-color: lighten(@table-hdr-brd-color, 12%);
@group-txt-color: #777;
@group-brd-color: @table-hdr-brd-color;


.data-grid-container {
    overflow-x: hidden;
    overflow-y: auto;
    position: relative;

    &.bordered {
        border: 1px solid @data-grid-border-color;
    }
}

table.data-grid {
    cursor: default;
    width: 100%;

    col.fit {
        width: 1px;
    }

    &.striped {
        > tbody > tr.even {
            background-color: @alter-bg-color;
        }
    }

    td {
        padding: 4px 6px;
        vertical-align: top;
        border-color: @table-brd-color;
```

```
    > i.ico, > a > i.ico {
        //float:left; /*for right aligned icons ??? */
        padding-right: 4px;
        font-size: inherit;

        &.ico-single {
            padding-right: 0;
            //float:none;
        }
    }
}

th {
    border-color: @table-hdr-brd-color;
    text-align: left !important;
}

td.cell-editable {
    padding: 0;

    .control-group {
        margin: -1px;

        .input-group {
            //border-style:none!important;
        }
    }
}

td.cell-validator {
    padding: 4px 4px;
    width: 23px;
}

colgroup > col.sorted {
    background-color: @sort-bg-color;
}

th {
    background-color: @dg-header-bg-color;
    color: @dg-header-txt-color;
    border-bottom: 1px solid @table-hdr-brd-color;
    vertical-align: baseline;
    //position: relative; // Firefox BUG: hide borders
    .noselect();

    .h-holder {
        position: relative;
        padding: 4px 6px;
    }

    &.sort {
        cursor: pointer;
        white-space: nowrap;

        &:hover {
            background-color: @dg-header-hover-color;
        }

        .h-holder:after {
            content: '';
            border: 5px solid transparent;
            margin-left: 2px;
        }
```

```
    }

    &.sort.desc, &.sort.asc {
        background-color: @dg-sort-column-bg-color;
    }

    &.sort.desc .h-holder:after {
        content: '';
        display: inline-block;
        border: 5px solid transparent;
        border-top-color: #999;
        margin-left: 2px;
        vertical-align: bottom;
    }

    &.sort.asc .h-holder:after {
        content: '';
        display: inline-block;
        border: 5px solid transparent;
        border-bottom-color: #999;
        margin-left: 2px;
        vertical-align: top;
    }
}

tr.active {
    background-color: @active-bg-color !important;
}

tr.row-details {
    &:hover {
        background-color: white !important;
    }

    td.details-cell {
        background-color: #fefcea;
        padding: 0;
    }

    .details-wrapper {
        padding: 4px 6px;
        border-top: 1px solid @table-brd-color;
        border-bottom: 1px solid @table-brd-color;
    }
}

td.danger, td.error, td.red {
    background-color: @dg-danger-bg-color !important;
}

td.warning, td.orange {
    background-color: @dg-warning-bg-color !important;
}

td.success, td.green {
    background-color: @dg-success-bg-color !important;
}

td.info, td.cyan {
    background-color: @dg-info-bg-color !important;
}

td.marker {
    padding: 0;
```

```
    min-width: 12px;
    height: 100%;
    position: relative;

    > div {
        //TODO: HACK ???
        position: absolute;
        left: 0;
        top: 0;
        bottom: 0;
        right: 0;
        //height:50%;
        border-left: 6px solid #ddd;
        background-color: @alter-bg-color;
    }

    > .warning, > .yellow {
        background-color: @dg-warning-bg-color;
        border-left-color: @dg-warning-brd-color;
    }

    > .danger, > .error, > .red {
        background-color: @dg-danger-bg-color;
        border-left-color: @dg-danger-brd-color;
    }

    > .success, > .green {
        background-color: @dg-success-bg-color;
        border-left-color: @dg-success-brd-color;
    }

    > .info, > .cyan {
        background-color: @dg-info-bg-color;
        border-left-color: @dg-info-brd-color;
    }
}

td.details-marker {
    padding: 0;
    padding: 4px;
    width: 22px;
    cursor: pointer;

    .ico {
        padding: 0;
    }
}

th.group-cell {
    white-space: nowrap;
    overflow: hidden;

    a {
        display: inline-block;
        padding: 3px 0;
        min-width: @group-padding;
        border: 1px solid transparent;
        color: @dg-header-txt-color;
        font-size: 85%;
        text-align: center;

        &:hover {
            background-color: @dg-header-hover-color;
            text-decoration: none;
```

```
        }
    }

    a + a {
        border-left-color: @table-hdr-brd-color;
    }
}

tr.group {
    td {
        background-color: @group-bg-color;
        border-bottom: 1px solid @group-brd-color;
        cursor: pointer;
        color: @group-txt-color;
        padding-left: 0;

        & .expmark:after {
            font-family: 'Bowtie';
            content: '\e9cd'; /*tree-collapsed*/
            width: 16px;
            display: inline-block;
            vertical-align: top;
            font-size: 16px;
        }

        & .expmark.expanded:after {
            content: '\e9cb'; /*tree-expanded*/
        }

        .grtitle {
            font-weight: bold;
        }

        .grcount {
            float: right;
            font-size: 85%;
            background-color: @dg-bage-bg-color;
            color: white;
            padding: 2px 4px;
            border-radius: 4px;
        }
    }

    &.lev-2 td {
        padding-left: @group-padding;
    }
}

td.group-marker {
    padding: 0;
    border-style: none;
}

tr.lev-2 .group-marker {
    //padding-left: @group-padding * 2 + @group-padding / 2;
}

tr.lev-3 .group-marker {
    //padding-left: @group-padding * 3 + @group-padding / 2;
}

tr:last-child > td {
    border-bottom: 1px solid @table-brd-color;
}
```

```
}


table.data-grid.compact {
    //TODO: may be .small-font ????
    font-size: .95rem;
    line-height: normal;

    td {
        padding: 2px 4px;
    }
}

table.data-grid > tbody {
    > tr.warning, > tr.orange {
        td {
            background-color: @dg-warning-bg-color;
        }
    }

    > tr.danger, > tr.error, > tr.red {
        td {
            background-color: @dg-danger-bg-color;
        }
    }

    > tr.success, > tr.green {
        td {
            background-color: @dg-success-bg-color;
        }
    }

    > tr.info, > tr.cyan {
        td {
            background-color: @dg-info-bg-color;
        }
    }

    > tr.bold {
        td {
            font-weight: @font-bold;
        }
    }
}

.data-grid-container.fixed-header {
    height: 100%;
    overflow: hidden;
    position: relative;
    min-height: 2rem + 1px;

    > .data-grid-body.fixed-header {
        margin-top: 2rem;
        overflow: hidden;
        overflow-y: auto;
        // 25px (header) + 27px(pager)
        height: ~"calc(100% - 2rem)";
        //border-bottom: 1px solid @table-hdr-brd-color;
        > table > thead {
            height: 0;

            > tr > th {
                border-top-style: none;
                border-bottom-style: none;
```

```
                height: 0;
                visibility: hidden;

                > .h-fill {
                    padding: 0 @table-cell-padding-h;
                    height: 0;
                    color: transparent;
                    line-height: 0 !important;
                    margin-top: -5px;
                    opacity: 0;
                    white-space: nowrap;
                }

                &.sort > .h-fill:after {
                    content: '';
                    border: 5px solid transparent;
                    margin-left: 2px;
                }

                > .h-holder {
                    position: absolute;
                    visibility: visible;
                    top: 0;
                    background-color: inherit;
                    width: 100%;
                    min-height: 2rem;
                    margin-left: -1px;
                    border-left: 1px solid @table-hdr-brd-color;
                    border-bottom: 1px solid @table-hdr-brd-color;
                    border-top: 1px solid @table-hdr-brd-color;
                }

                &:first-child > .h-holder {
                    border-left-style:none;
                }
            }
        }
    }

    &.bordered {
        > .data-grid-body.fixed-header {
            .h-holder {
                border-top-style: none !important;
            }

            th:first-child {
                border-left-style: none !important;
            }

            tbody > tr > td:first-child {
                border-left-style: none !important;
            }

            tbody > tr > td:last-child {
                border-right-style: none !important;
            }

            tbody > tr:last-child > td {
                //border-bottom-style: none !important;
            }
        }
    }
}
```

Файл main.less

```
html {
    font-family: @font-family;
    font-size: @font-size;
    -ms-text-size-adjust: 100%;
    -webkit-text-size-adjust: 100%;
    color: @text-color;
    box-sizing: border-box;
    padding: 0;
}

input, textarea, select, button {
    font-family: @font-family;
}

body {
    overflow: hidden;
    height: 100%;
    width: 100%;
    padding: 0;
    background-color: white;
    .noselect();
}

[v-cloak] {
    display: none;
}

button {
    font-size: @font-size;
    border: 1px solid transparent;
    cursor: pointer;

    &:focus {
        outline-style: none;
    }
}


hr {
    margin: 0.3em 0;
    border: 1px solid #ccc;
    border-bottom-style: none;
}

header, .header {
    position: absolute;
    left: 0;
    top: 0;
    right: 0;
    height: @header-height;
    padding: 0 4px 0 8px;
    background-color: @brand-bk_color;
    color: white;
    display: flex;
    align-content: flex-start;
    align-items: center;

    .h-block {
        white-space: nowrap;

        a.app-title {
            color: inherit;
```

```
            font-size: 18px;

            .ico {
                font-size: inherit;
                margin-right: 6px;
            }

            &:hover {
                text-decoration: none;
                color: inherit;
            }
        }

        .app-subtitle {
            padding-left: 6px;
        }
    }

    > a.nav-admin {
        color: inherit;
        padding: 4px 6px;

        &:hover {
            background-color: @brand-hover-color;
        }
    }

    > .dropdown {
        display: inline-block;

        .menu {
            margin-top: -1px;
            min-width: 185px;
        }
    }

    .btn {
        background-color: transparent;
        color: white;
        padding: 4px;

        &:hover {
            background-color: @brand-hover-color;
        }
    }
}


@side-bar-width: 250px;
@side-bar-gap: 6px;
@side-bar-collapsed-width: 20px;

.content-view {
    position: absolute;
    left: 0;
    right: 0;
    top: 0;
    bottom: @footer-height;
}

.content-view.partial-page {
    left: @side-bar-width + @side-bar-gap;
    top: 70px //64px + 6px;
    //background-color:aliceblue;
```

```
}

.side-bar-collapsed .content-view.partial-page {
    left: @side-bar-collapsed-width + @side-bar-gap;
}

.content-view.full-page {
    top: 64px;
}

.content-view.full-view {
    top: 32px;
}

.include {
    width: 100%;
    height: 100%;
    position: relative;
    white-space: normal; // override
}

.include.loading {
    > div {
        display: none;
    }
}

@import "Nav.less";
@import "Page.less";
@import "Splitter.less";
@import "Taskpad.less";


.side-bar {
    position: absolute;
    left: 0;
    top: @header-height + @navbar-height;
    margin: @side-bar-margin;
    bottom: @footer-height;
    width: @side-bar-width;
    padding: 16px 0 0 0;
    border-right: 1px solid @side-bar-brd-color;
    background-color: @side-bar-bg-color;

    .side-bar-body, .tree-view {
        height: 100%;
    }

    .side-bar-title {
        background-color: @side-bar-collapsed-bg-color;
        white-space: nowrap;
        height: 100%;
        cursor: pointer;
        font-size: 14px;
        padding-left: 2px;

        .side-bar-label {
            display: block;
            padding-left: 10px;
            transform: rotate(90deg);
        }
    }

    .collapse-handle {
```

```
        position: absolute;
        top: 0;
        right: 0;
        width: auto;
        padding: 2px 4px;
        color: @default-brd-color;
        font-size: 110%;

        &:hover {
            color: @link-hover-color;
            text-decoration: none;
            background-color: @side-bar-collapsed-bg-color;
        }

        &:before {
            content: "\e9a3"; // chevron-left
        }
    }

    &.collapsed {
        width: @side-bar-collapsed-width;
        background-color: @side-bar-collapsed-bg-color;

        .collapse-handle {
            padding-right: 6px;

            &:before {
                content: "\e9a5"; // chevron-right
            }
        }
    }
}

footer, .footer {
    position: absolute;
    display:flex;
    align-items:center;
    left: 0;
    bottom: 0;
    right: 0;
    height: @footer-height;
    background-color: @footer-bg-color;
    border-top: 1px solid @default-brd-color;
    z-index: @footer-index;
    padding: 0 2px 0 8px;
    font-size: .86rem;
    line-height:1.5rem;
    text-transform:lowercase;

    .divider {
        flex-grow:2;
    }

    .debug-btn {
        display:inline;
        margin-left:0.5rem;
    }

    .menu {
        margin-bottom: -2px;
        font-size: @font-size;
    }

    button {
```

```
        background-color: @danger-btn-color;
        font-size: inherit;
        padding: 0 6px 1px 6px;
        color: white;
        text-transform: lowercase;
        vertical-align: baseline;

        &:focus {
            outline-style: none;
        }
    }

    .spinner {
        display: inline-block;
        box-sizing: content-box;
        width: 8px;
        height: 8px;
        margin-right: 2px;
        background-color: transparent;
        border-radius: 50%;
        border: 3px solid #ccc;
        vertical-align: -2px;

        &.active {
            border-color: @link-hover-color;
            border-top-color: #ccc;
            animation: spin 1.5s linear infinite;
        }
    }
}


@keyframes spin {
    0% {
        transform: rotate(0deg);
    }

    100% {
        transform: rotate(360deg);
    }
}


input, select, textarea {
    border-style: none;
    font-size: @font-size;
    color: @text-color;

    &:focus {
        outline-style: none;
    }
}

textarea {
    padding: 2px @input-padding;
}

input, select {
    height: @control-height;
    padding: 0 @input-padding;
}

/* hide for focused */
input::-moz-placeholder {
```

```
    color: @placeholder-color;
}

input:focus::-moz-placeholder {
    color: transparent;
}

input:-ms-input-placeholder {
    color: @placeholder-color;
}

input:focus:-ms-input-placeholder {
    color: transparent;
}

input::-webkit-input-placeholder {
    color: @placeholder-color;
}

input:focus::-webkit-input-placeholder {
    color: transparent;
}

select {
    padding: 0 @select-padding;
}

textarea {
    resize: none;
}

a {
    color: @link-color;
    text-decoration: none;
    cursor: pointer;

    &[disabled] {
        opacity: 0.5;
        cursor: not-allowed;
        //pointer-events: none; // disable click
    }

    &:hover {
        color: @link-hover-color;
        text-decoration: none;
    }

    &:focus {
        color: @link-hover-color;
        outline-style: none;
    }

    &:active {
        color: @link-hover-color;
    }
}

pre {
    font-family: @monospace-font;
    font-size: 0.85em;
    background-color: #f8f8f8;
    padding: 6px;
    white-space: pre-wrap;
    word-break: break-all;
```

```
    word-wrap: break-word;
    user-select: text;
}

pre.a2-code {
    overflow: auto;
    max-height: 300px;
    margin: 0;
}


@import "Control.less";
@import "Button.less";
@import "DatePicker.less";
@import "Selector.less";
@import "DataGrid.less";
@import "ToolBar.less";
@import "TreeView.less";
@import "Tab.less";
@import "List.less";
@import "Dialog.less";
@import "Table.less";
@import "PropertyGrid.less";
@import "CheckBox.less";
@import "Pager.less";
@import "DropDown.less";
@import "Popover.less";
@import "Grid.less";
@import "Layout.less";

@import "Sheet.less";

.app-exception {
    margin: 50px auto;
    width: 600px;
    height: auto;
    text-align: center;
    z-index: auto; //@exception-index;
    position: relative;
    padding: 0;
    background-color: @danger-bg-color;
    color: @danger-txt-color;
    border: 1px solid @danger-brd-color;
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.2);

    .message {
        padding: 15px 10px;
        white-space: pre-line;
        font-weight: bold;
    }

    .stack-trace {
        border-top: 1px solid @danger-txt-color;
        background-color: lighten(@danger-bg-color, 5%);
        overflow: auto;
        padding: 10px;
        max-height: 200px;
        font-size: 92%;
    }
}

.load-indicator {
    position: absolute;
    left: 0;
```

```
    right: 0;
    top: @header-height - 2px;
    height: 4px;
    width: 100%;
    overflow: hidden;
    background-color: darken(@navbar-bg-color, 10%);

    &:before {
        content: '';
        display: block;
        position: relative;
        left: 0;
        width: 50%;
        height: 4px;
        background-color: gold;
        animation: loading 2s linear infinite;
    }
}

@keyframes loading {
    from {
        left: 0;
    }

    to {
        left: 100%;
    }
}

h2, .h2 {
    font-size: 1.6em;
}

h3, .h3 {
    font-size: 1.2em;
}

h4, .h4 {
    font-size: 1em;
}

h5, .h5 {
    font-size: 0.85em;
}

.a2-header {
    margin: 0.5em 0;
    color: #777;
    font-weight: bold;
    font-family: inherit;
}

.file-upload {
    display: block;
    width: 100px;
    height: 100px;
    padding: 0;
    background-color: lemonchiffon;
    border: 2px dotted silver;

    &.hover {
        border-color: gold;
    }
```

```
    input {
        width: 100%;
        height: 100%;
        font-size: 1px;
        opacity: 0;
    }
}

.badge {
    display: inline-block;
    background-color: @dark-bk-color;
    color: white;
    padding: .25rem .4rem;
    margin-left: 0.4rem;
    font-size: 85%;
    font-weight: @font-bold;
    line-height: 1;
    text-align: center;
    white-space: nowrap;
    vertical-align: baseline;
    border-radius: .25rem;
}

.xaml-exception {
    padding: 6px;
    display: inline-block;
    background-color: @danger-bg-color;
    border: 1px solid @danger-brd-color;
    color: @danger-txt-color;
}

.not-supported {
    width: 400px;
    margin: 20px auto;
    padding: 20px;
    text-align: center;
    background-color: @danger-bg-color;
    color: @danger-txt-color;
    border: 1px solid @danger-brd-color;
}

.a2-graphics {
    display: block;
    width: 100%;
}
```

Файл layout.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta name="description" content="A2:Web" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="format-detection" content="telephone=no" />
    <meta name="rootUrl" content="$(RootUrl)" />
    <meta name="helpUrl" content="$(HelpUrl)" />
    <meta name="google" content="notranslate" />
    <title></title>
    <link href="/css/$(Theme).min.css?v=$(Build)" rel="stylesheet" />
    <link rel="shortcut icon" href="/favicon.ico" type="image/x-icon" />
    <link rel="icon" href="/favicon.ico" type="image/x-icon" />
</head>
<body>
    <div id="shell" class="shell">
        <header class="header">
            <div class="h-block">
                <!--<i class="bowtie-user"></i>-->
                <a class="app-title" href='/' @click.prevent="root" v-text="title" tabindex="-
1"></a>
                <span class="app-subtitle" v-text="subtitle"></span>
            </div>
            <div class="aligner"></div>
            <a class="nav-admin" v-if="userIsAdmin" href="/admin/" tabindex="-1"><i class="ico
ico-gear-outline"></i></a>
            <div class="dropdown dir-down" v-dropdown>
                <button class="btn" toggle><i class="ico ico-user"></i> $(PersonName)<span
class="caret"></span></button>
                <div class="dropdown-menu menu down-right">
                    <a @click.prevent="profile" class="dropdown-item" tabindex="-1"><i
class="ico ico-user"></i> Профиль</a>
                    <a @click.prevent="changePassword" class="dropdown-item" tabindex="-1"><i
class="ico ico-lock"></i> Сменить пароль</a>
                    <template v-if="userIsAdmin">
                        <div class="divider"></div>
                        <a @click.prevent="changeUser" class="dropdown-item" tabindex="-1"><i
class="ico ico-switch"></i> Сменить пользователя</a>
                    </template>
                    <div class="divider"></div>
                    <form id="logoutForm" method="post" action="/account/logoff">
                        <a href="javascript:document.getElementById('logoutForm').submit()"
tabindex="-1" class="dropdown-item"><i class="ico ico-exit"></i> @[Quit]</a>
                    </form>
                </div>
            </div>
        </header>
        <a2-main-view :menu="menu"></a2-main-view>
        <a2-debug :model-stack="modelStack" :model-visible="debugShowModel"
                  :trace-visible="debugShowTrace" :counter="dataCounter"
:close="debugClose"></a2-debug>
        <footer class="footer">
            <div :class="{spinner: true, active:processing}"></div>

            <span class="divider"></span>

            <a href="/app/about" @click.prevent="about" tabindex="-1">@[About]</a>


            <span class="divider"></span>
```

```
            @[Version] 
            <span v-text="version"></span>
            <div class="debug-btn dropdown dir-up" v-dropdown>
                <button class="btn btn-debug" toggle>@[Debug]<span class="caret
up"></span></button>
                <div class="dropdown-menu menu up-right">
                    <a @click.prevent="debugModel" class="dropdown-item" tabindex="-1"><i
class="ico ico-database"></i> Модель данных</a>
                    <a @click.prevent="debugTrace" class="dropdown-item" tabindex="-1"><i
class="ico ico-chart-stacked-line"></i> Профилирование</a>
                    <div class="divider"></div>
                    <!--
            <a @click.prevent="debugOptions" class="dropdown-item"><i class="ico ico-
wrench"></i> Настройка</a>
            -->
                    <label class="checkbox">
                        <input type="checkbox" v-model="traceEnabled" />
                        <span>Трассировка</span>
                    </label>
                </div>
            </div>
        </footer>
        <!--
        <a2-debug v-if="debugVisible"></a2-debug>
        -->
    </div>
    <script type="text/javascript" src="/scripts/vue.js?v=$(Build)"></script>
    <script type="text/javascript" src="/scripts/d3.min.js?v=$(Build)"></script>
    <script type="text/javascript" src="/scripts/main.js?v=$(Build)"></script>
    <script type="text/javascript" src="/shell/script"></script>
</body>
</html>
```