Selective EXPORT

Two new overloads for the *export* method were added to *BlockServices* to allow enhanced export customization.

The first one allows to specify the list of items to export (*itemNames*) of a block (*model*).

The second provides the same functionality but for a set of blocks (*models*).

New BlockServices.export overloads

```
boolean export(IBusinessObject model, List<String> itemNames, Export
export, Map<String, Object> options, boolean fullExport, boolean
includeHeaders);
boolean export(Map<IBusinessObject, List<String>> models, Export export,
Map<String, Object> options, boolean fullExport, boolean includeHeaders);
```

A new entity (*IBusinessObjectItem)* was added in order to contain the state of each business object item in each user session.

The *Exportable* property is inherited from the manager configuration (IBusinessObjectItemConfiguration) and is enabled by default.

Note that the default implementation (*BusinessObjectItem)* only allows to modify the *Exportable* property if it was not explicitly disabled in manager configuration (with exportable="false").

However, the final decision on whether or not an item is exported (among others) is up to the *IExportDataProvider*.

By default the *ExportDataProvider* is used, which only allows export if the item has the *Exportable* property enabled, as can be seen in the following code excerpt:

ExportDataProvider.isItemExportable implementation

```
public boolean isItemExportable(IBusinessObject model, IBusinessObjectItem
item)
{
    if (item.getName().equals(DAOConfiguration.MODEL_ROWID))
        return false;

    if (!item.isExportable())
        return false;

    try
    {
        getExportableItem(model, item.getConfig());
    }
    catch (ExportRestrictedDataAccessException e)
    {
        return false;
    }

    return true;
}
```

In order to modify this behavior (among others) just create a new IExportDataProvider and configure it in AppSupportLibSettings:

IExportDataProvider configuration in appsupportlib.config.xml

```
<bean id="appSupportLibSettings"
class="morphis.foundations.core.appsupportlib.configuration.AppSupportLibS
ettings" init-method="initialize">

    <property name="exportDataProvider">
        <bean class=" com.example.CustomExportDataProvider"/>
    </property>

</bean>
```

Usage Examples

Export items

Example implementation of a generic action in a *TaskController* that receives from the client the list of items to export from the current block.

EXPORT-ITEMS example

```
@ActionTrigger(action="EXPORT-ITEMS")
public void exportItems(String export, String items)
{
    IBusinessObject model = getCurrentBlockController().getModel();
    List<String> itemNames = Arrays.asList(
StringUtils.tokenizeToStringArray(items, ",") );

    Map<String,Object> options = null;
    boolean fullExport = false;
    boolean includeHeaders = true;
    BlockServices.export(model, itemNames, Export.valueOf(export), options,
fullExport, includeHeaders);
}
```

Example of an EXPORT-ITEMS action call in Frames. This will export the specified items unless they have the property *Exportable* disabled. Row identifiers and unknown items are also ignored.

EXPORT-ITEMS usage example

```
Frames.Application.execute({
 name: 'EXPORT-ITEMS',
 params: [
  {name: 'export', type: 'string', value: 'Excel'},
  {name: 'columns', type: 'string', value: 'LASTNAME, JOB, SAL, HIREDATE,
F2N_ROWID, ROWID,    __INVALID_ITEM__'}
 ]
}, Frames.Application.task, undefined, undefined, undefined, false);
```

Set Exportable Items

Example implementation of a generic action in a *TaskController* that receives from the client the list of exportable items of the current block.

SET-EXPORTABLE-ITEMS example

```
@ActionTrigger(action="SET-EXPORTABLE-ITEMS")
public void setExportableItems(String items)
{
    IBusinessObject model = getCurrentBlockController().getModel();
    List<String> itemsToExport = Arrays.asList(
StringUtils.tokenizeToStringArray(items, ",") );

    for (IBusinessObjectItem item : model.getItems())
        item.setExportable( itemsToExport.contains(item.getName()) );
}
```

Example of a SET-EXPORTABLE-ITEMS action call in Frames. This will set the *Exportable* property enabled for the specified items (unless explicitly disabled in manager configuration) and disabled for all other items.

SET-EXPORTABLE-ITEMS usage example

```
Frames.Application.execute({
 name: 'SET-EXPORTABLE-ITEMS',
 params: [
  {name: 'columns', type: 'string', value: 'LASTNAME, JOB, SAL'}
 ]
}, Frames.Application.task, undefined, undefined, undefined, false);
```

Export blocks

Example implementation of a generic action in a *TaskController* that receives from the client a map of blocks and items to export.

EXPORT-BLOCKS example

```
 @ActionTrigger(action="EXPORT-BLOCKS")
public void exportBlocks(String export, String blocks)
{
    @SuppressWarnings("unchecked")
    Map<String,List<String>> blockMap = (Map<String,List<String>>) new
Gson().fromJson(blocks, Map.class);
    Map<IBusinessObject,List<String>> modelMap = new LinkedHashMap<>();
    for (String block : blockMap.keySet())
    {
        IBlockController bc = getBlockController(block);
        if (bc != null && bc.getModel() != null)
            modelMap.put(bc.getModel(), blockMap.get(block));
    }

    Map<String,Object> options = null;
    boolean fullExport = false;
    boolean includeHeaders = true;
    BlockServices.export(modelMap, Export.valueOf(export), options,
fullExport, includeHeaders);
}
```

Example of an EXPORT-BLOCKS action call in Frames. will export the specified blocks and items unless they have the property *Exportable* disab led. Row identifiers and unknown items are also ignored.

EXPORT-BLOCKS usage example

```
Frames.Application.execute({
 name: 'EXPORT-BLOCKS',
 params: [
  {name: 'export', type: 'string', value: 'Excel'},
  {name: 'models', type: 'string', value: '{ \
   \'DEPT\' : [\'DEPTNO\', \'DNAME\', \'RESPONSIBLE\'], \
   \'EMP\': [\'LASTNAME\', \'JOB\', \'SAL\', \'HIREDATE\']}'
  }
 ]
},
Frames.Application.task, undefined, undefined, undefined, false);
```