Additional Interfaces

Some additional interfaces are available which allow you to modify the binding behavior between a page and its adapter object. The interfaces are available via `com.softwareag.cis.server.IInteractionSessionMgr` which represents a general interface to the Application Designer runtime.

You receive an instance of `IInteractionSessionMgt` in the following way:

```
...
...
IInteractionSessionMgr iism = InteractionSessionMgrFactory.getInteractionSessionMgr();
...
...
```

This chapter covers the following topics:

- Extending the Set of Simple Data Types
- Avoid the Getting of Certain Simple Data Type Properties
- Exchanging Objects by Converter Objects

Extending the Set of Simple Data Types

As described previously in this part, Application Designer collects all the properties of an adapter object (and its contained objects) when collecting the data in order to respond to the browser client.

The way Application Designer collects the data for a certain object is:

- Application Designer collects all the properties that represent simple data types (int, float, String, BigDecimal, CDate, etc.; see *Appendix C - Data Types to be Used by Adapter Properties*).
- Application Designer investigates all properties that are non-simple datatypes and that are part of the access path of a certain page.

Somtimes you want to add a certain class to be managed as "Simple Datatype Class", i.e. Application Designer will not treat objects of this class as non-simple objects but will treat them as simple objects.

Simple objects have to provide a class implementation that

- provides a constructor in which the value is passed as a string object, and
- provides a `toString()` method to get the String representation of the contained value.

Example of a valid class:

```
public class ExtendedString
{
    String m_value;
    public ExtendedString(String value)
    {
        m_value = value;
    }
    public String toString()
    {
        return m_value;
    }
    ...
    ...
}
```

The class is registered by using the method `IInteractionSessionMgr.registerPropertyAccessSimpleDatatypeExtension()`:

```
IInteractionSessionMgr iism;
iism = InteractionSessionMgrFactory.getInteractionSessionMgr();
iism.registerPropertyAccessSimpleDatatypeExtension(ExtendedString.class);
```

Now having an adapter object (or follow-on object such as grid item) providing a property of type `ExtendedString`, Application Designer will not drill down the object but will use the object's `toString()` method to get its value and will use the object's constructor to pass new values to the application.

Avoid the Getting of Certain Simple Data Type Properties

In the previous sections, the general rule was explained: if Application Designer investigates an object during the get-data phase, then

- it takes all simple data type properties, and
- it takes those complex data type properties that are required by the corresponding page.

There is one possibility to fine-control the getting of simple data type properties: Every object that is investigated by Application Designer during the get phase (e.g. the adapter object) can implement the interface `com.softwareag.cis.server.IControlPropertyAccess`. The interface is defined as follows:

```
public interface IControlPropertyAccess
{
    public String[] findPropertiesNotToBeCollected();
}
```

When the interface is implemented, the get methods that are passed back by the `findPropertiesNotToBeCollected()` method are not processed.

Note that the method is called once per class - the first time Application Designer interacts with an object. You cannot tell Application Designer by this interface to sometimes use the property and sometimes not.

Exchanging Objects by Converter Objects

When Application Designer is accessing properties that are non-simple data type objects, there is the possibility to exchange the object and tell Application Designer to use a converter object instead.

The interfaces are:

- With `IInteractionSessionMgr.registerPropertyAccessConverter(Class forClass, IPropertyAccess Converter converter)` you can register a class (parameter `converter`) that is used as converter for another class (parameter `forClass`).
- The converter class itself must support the interface `IPropertyAccessConverter` that looks as follows:

  ```
  public interface IPropertyAccessConverter
  {
      public Object getConvertedObject(Object propertyValue);
  }
  ```

For more details, see the JavaDoc API documentation.