Application Designer

Appendices

Version 9.1.4

October 2021

ADABAS & NATURAL

This document applies to Application Designer Version 9.1.4 and all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: CIT-APPENDICES-914-20210918

Table of Contents

Preface

The following appendices are available:

Appendix A - Call Sequence for Adapter	Describes how an incoming request by the browser client is processed inside an adapter.
Appendix B - Usage of Methods Inherited from the Adapter Class	Gives information about adapter classes and how to use them.
Appendix C - Data Types to be Used by Adapter Properties	Describes the various data types that can be used by adapter properties.
Appendix D - Class Loader Concepts	Gives information about class loader management.
Appendix E - StartCISPage Servlet	Describes the StartCISPage servlet that is used to open intelligent HTML pages.

1 About this Documentation

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
`Monospace font`	Identifies service names and locations in the format *`folder.subfolder.service`*, APIs, Java classes, methods, properties.
`Italic`	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at **https://documentation.softwareag.com**.

Software AG Empower Product Support Website

If you do not yet have an account for Empower, send an email to empower@softwareag.com with your name, company, and company email address and request an account.

Once you have an account, you can open Support Incidents online via the eService section of Empower at **https://empower.softwareag.com/**.

You can find product information on the Software AG Empower Product Support website at **https://empower.softwareag.com**.

To submit feature/enhancement requests, get information about product availability, and download products, go to **Products**.

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the **Knowledge Center**.

If you have any questions, you can find a local or toll-free number for your country in our Global Support Contact Directory at **https://empower.softwareag.com/public_directory.aspx** and give us a call.

Software AG Tech Community

You can find documentation and other technical information on the Software AG Tech Community website at **https://techcommunity.softwareag.com**. You can:

- Access product documentation, if you have Tech Community credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

2 Appendix A - Call Sequence for Adapter

This chapter describes how an incoming request by the browser client is processed inside an adapter. The request contains all the changes of properties that have been made at client side.

Normal Call Sequence

- `init()`
  This method is called only once - when creating the adapter inside a subsession. Before calling this method, Application Designer makes sure that the adapter instance is properly registered inside the Application Designer environment. Therefore - for example - you have access to the session management: use the `findSessionContext()` or `findSubSessionContext()` method in order to look for some values inside the `init()` method. It is not possible to use the `find...SessionContext()` methods inside the constructor of an adapter - since the session is not yet assigned to the adapter instance.

  When navigating between pages (using the `switchToPage()` or `openPopupPage()` method), the corresponding adapter objects are only created once. For example, if you navigate from page "A" to page "B" and back to page "A", the adapter of page "A" does not change. The `init()` method is only called once - at the time the adapter is instanciated.
- `activate(...)`
  This method is implemented by the `Adapter` class already. You only need to overwrite this method if you want to passivate the state between requests. In this case, you can activate this state inside your implemented method of your adapter class. If you use the adapter class to co-operate, for example, with components running in a container of an application server, you should synchronize the state passivation with the container's passivation.
- `reactOnDataTransferStart()`
  This method is called when the transfer of the changed properties starts. You can initialize some internal members at this time. If you overwrite this method, do not forget to include the method of the super-class (`Adapter.reactOnDataTransferStart()`) into your method implementation!
- `setXxx(), setYyy(),...`
  Now, the set methods of the changed properties of the browser client are transferred. It is very important that your implemented set methods never cause an exception or an error.
- `reactOnDataTransferEnd()`
  This method is called after setting the changed properties. Use this method to perform operations you always want to execute when processing a request.
- `invoke()`**-Method**
  If the request has a method call inside, the method is invoked now.
- `processAsDefault()`
  If the request has no method call, this standard method is called.

- `reactOnDataCollectionStart()`
  This method is called when the transfer of adapter properties starts. Use this method, for example, for performance improvements during the following get methods, for example, by building temporary objects.
- `getXxx(),getYyy()`
  All get methods of the adapter - including array elements which may be passed back by - are called.
- `reactOnDataColletionEnd()`
  This method is called when data collection is finished. Temporary objects - which you may have created for performance reasons - can be released for garbage collection now.
- `passivate(...)`
  This method is the counterpart of the activate method.

Call Sequence when a Subsession is Destroyed

- `endProcess()`
  This method is called inside the adapter if the user decides to terminate the subsession. For example, in the Application Designer workplace environment, this method is called whenever the user chooses the close button of a page.

You can deny closing a subsession in your implemented method:

```
public class ABCAdapter
    extends com.softwareag.cis.server.Adapter
{
    ...
    ...
    public void endProcess()
    {
        // veto the endProcess in case of unsaved data
        if (changedDataNotSaved == true)
        {
            this.outputMessage("E","Please save data first");
            return;
        }
        // close subsession
        super.endProcess();
    }
}
```

Call Sequence when a Session is Destroyed

If a session is removed from Application Designer - for example, if the user closes the browser or if a system administrator removes the session - the adapter instances are informed in the following way:

- `destroy()`
  In your implemented method, clean up all resources bound to your adapter instance. You cannot deny the destroying of the session - but you can react.

Error/ Runtime Exceptions

Error and runtime exceptions occurring during the adapter request processing may be handled centrally inside your adapter. For more details, see *Binding between Page and Adapter* in the *Special Development Topics*.

Pay Attention when Overwriting

The methods named above are already implemented with default behavior inside the class `com.softwareag.cis.server.Adapter`. Pay attention when overwriting these methods inside your adapter and always include the super-class's processing into your own implementation. The first statement inside your implementation should call the super-class method:

```
public class ABCAdapter
    extends com.softwareag.cis.server.Adapter
{
    ...
    ...
    public void reactOnDataTransferStart()
    {
        super.reactOnDataTransferStart();
        // now own implementation
        ...
        ...
    }
}
```

3 Appendix B - Usage of Methods Inherited from the Adapter Class

Inside the Application Designer management, adapters have to provide a defined interface to be managed correctly by the system. This interface is declared by `com.softwareag.cis.Server.IAdapter`. In order to have a high level of comfort during developing adapters, you should derive your adapter classes from the super-class `com.softwareag.cis.Server.Adapter`. This class already provides some useful methods.

Access to Lookup Session Context

As you know, session management defines sessions (corresponding to one browser instance) and subsessions (corresponding to one process inside the Application Designer workplace). There is the possibility to bind and look for parameters on both levels:

- `Adapter.findSessionContext()` - returns the context which is on top of all subsessions. All adapters inside one session refer to the same session context.
- `Adapter.findSubSessionContext()` - returns the context which is held per subsession. Only adapters - belonging to the same subsession - share this context.

The result is a context supporting the interface `com.softwareag.cis.context.ILookupContext`. This interface provides two important methods:

```
public Object lookup(String s, boolean reactWithErrorIfNotExist);
public void bind(String s, Object o);
```

The session context is used, for example, to refer to the current user who is logged in, the chosen language, etc. The subsession context is used to share data inside a subsession.

Do not use the context as global variable buffers in a very intensive way. It will end up in programs relying on a lot of context information to be available - and sooner or later no one knows what has to be in the context when starting the program.

Via the methods

- `Adapter.findSessionId()`
- `Adapter.findSubsessionId()`

you can access the internally used representations of session ID and subsession ID.

Access to Application Designer Session Context

Application Designer uses its own lookup session management in order to store information of a session. You can access and manipulate this information by calling your adapter's method:

- `Adapter.findCISessionContext()` - returns a concrete session context object.

Inside the session context, the following parameters are kept:

- date format
- time format
- language
- style
- decimal separator
- and other information.

Have a look at the JavaDoc API documentation for more details.

Access to other Adapters

Access other adapters inside the same subsession by the methods:

- `Adapter.findAdapter(class)` - returns the adapter instance for a given class. Method `init()` is already called when passing back the instance - but only if the adapter was not used before.

Use this method before navigating between pages in order to prepare the adapter that will be used by the next page.

Error Output

You can display error messages inside the status bar (if it is defined in the page layout) by using the methods:

- `outputMessage(String, String (, String))`

First, pass a string for the type of message. This is needed to display a corresponding icon inside the status bar. There are constants defined inside the Adapter for specifying the type:

- `Adapter.MT_ERROR`

- `Adapter.MT_WARNING`
- `Adapter.MT_SUCCESS`

The second string is the message being shown.

The third string - which is optional - is the long text description of the message. It becomes visible by a dialog if the user clicks with the mouse on the message. If you do not specify a long description, the normal message is used.

Page Navigation

Navigate to a page by using the method:

- `switchToPage(String pageName)`

The "pageName" is the URL - either relative or absolute - of the next page.

Opening of Pop-up Dialogs

You can open a page inside a pop-up dialog by using the method:

- `openPopup(String pageName).`

The "pageName" is the URL - either relative or absolute - of the page that is displayed inside the dialog.

You can specify pop-up parameters of the pop-up you open with `openPopup()` by using the methods:

- `setPopupTitle(String title)`
- `setPopupPageFeatures(String pageFeatures)`

Frame Communication

There are various methods to communicate to other frames:

- `openPageInTarget`
- `openCISPageInTarget`
- `invokeMethodInTarget`

- `refreshTarget`
- `sizeTarget`

Closing of a Page

The default method used for closing a page is `endProcess()`. It is provided by the `Adapter` class. The tasks performed by the `endProcess()` method are:

- The current subsession is closed and de-registered inside the session management.
- The current page is de-registered from the workplace management - if it was registered before.

Calling the `endProcess()` method ensures that all memory resources are released for the corresponding subsession.

The `endProcess()` method is called by clicking inside the page on the close icon at the top right corner of the page. You can also call it directly inside an adapter, e.g. if you want to close the subsession as reaction to the user's entered data.

Multi Language Management

You can access the multi language management using the methods:

- `replaceLiteral(String application, String textid)`
- `replaceLiteral(String application, String textid, String param1)`
- `replaceLiteral(String application, String textid, String param1, String param2)`
- `replaceLiteral(String application, String textid, String param1, String param2, Stirng param3)`

The application is the name for the abbreviation of a defined application area for which literals are defined. In the file-based multi language management, it represents the name of a CSV file that holds the text identified by a text ID.

4 Appendix C - Data Types to be Used by Adapter Properties

The Application Designer management is very flexible by allowing various data types for properties of an adapter.

Supported Data Types

- String
- int, long, short, byte
- float, double
- BigDecimal
- boolean
- CDate
- CTime
- CTimeStamp

Data Types for Managing Date and Time

The `java.util.Time` class is very powerful, but also very complex to use for business applications. Therefore, three classes are introduced to deal with date and time:

- `com.softwareag.cis.util.CDate`
- `com.softwareag.cis.util.CTime`
- `com.softwareag.cis.util.CTimeStamp`

See the JavaDoc documentation for further details.

Dates and times are transferred as strings between Application Designer and the intelligent HTML page:

- YYYYMMDD format for dates.
- HHMMSS format for times.
- YYYYMMDDHHMMSSMMM format for timestamps.

The interpretation and formatting of these strings to valid formats is done automatically.

5 Appendix D - Class Loader Concepts

An explicit class loader management was introduced to support the following scenarios:

- Classes are automatically found in the context of Application Designer without specifying a `CLASSPATH` variable.
- Classes can be stored inside an application project directory - separated from other application projects.
- During development time, easily run new pages together with the latest classes without restarting the server.

This chapter explains the class loader concepts used inside Application Designer.

Design Time - Runtime

The class loader concepts are designed to simplify the development of pages and their logical representations on the server side: adapters.

At runtime, they should only be used if you are not running in a cluster - i.e. if you do not distribute your application server on multiple nodes. When running in a cluster, classes should be located exactly there, where the application server specifications allow them to be located. Inside the Application Designer configuration, you can select which mode you are running in - for details, see *Design Time Mode and Runtime Mode* in the *Configuration* documentation.

After explaining the class loader concepts in this chapter, at the end we explain what to do in order to change a design time environment into a runtime environment.

Class Loader Hierarchy

Application Designer runs as a web application inside a servlet engine - by default, the Tomcat servlet engine is used. The class loader used by the servlet engine is called “web application loader” in the following text.

The Application Designer environment itself is running in the context of the web application loader. This class loader is looking for classes as specified by the servlet engine. Therefore the Application Designer runtime must be accessable by this class loader. For Tomcat, this is achieved by placing the *cis.jar* file inside the *<installdir>/tomcat/webapps/ROOT/WEB-INF/lib* directory.

The following topics are covered below:

Application Class Loader

The application classes (adapter classes) are loaded by the class loader management of Application Designer. This class loader looks for Java classes as follows:

- All *.class* files inside the directory:

  <webapp>/softwareag/appclasses/classes

- All *.jar* files inside the directory:

  <webapp>/softwareag/appclasses/lib

- All *.class* files inside any application project under the directory:

  <webapp>/<project>/appclasses/classes

- All *.jar* files inside any application project under the directory:

  /<webapp>/<project>/appclasses/lib

- All classes that are referenced in the classpath extension that can be defined in the Application Designer configuration (*cisconfig.xml*).

Unlike normal class loader hierarchies, the application class loader always tries to resolve a class inside its application directories first. Only if the class is not found, the parent class loader is called - the web application loader. The benefit is that application classes are totally separated from the servlet engine classes - e.g. by using XML parser libraries. You are not bound to the parser delivered with the servlet engine.

Inside the Application Designer session management, a session is bound to an application class loader instance. Therefore the application class loader - which was instanciated when the session was created - is kept in the session during its whole life cycle. All objects created inside this session use this instance of the class loader.

In case of changing classes inside the *softwareag/appclasses* or the corresponding application-project subdirectories, you can force to create a new class loader used in all sessions which are created afterwards. This means, that you can upgrade your system without disturbing running sessions. Old sessions are still using their old classes; new sessions are using new classes.

The creation of a new instance of a class loader is triggered inside the monitoring tool. See *Monitoring* in the *Development Workplace* documentation.

By choosing the button **Use latest Version of Applications for new Sessions**, a new class loader instance is generated.

A new class loader instance can also be created during development inside the Layout Painter. See also the "Hello World!" example in the *First Steps* and its section *If you Change the Adapter*.

Initialisation of Your Application

Every time a new instance of a class loader generated, the initialisation process of your application is also performed. This guarantees that, for example, all static variables you may use internally can be correctly initialised by your initialisation procedure.

The initialisation of applications is described in the *Becoming a Member of the Startup Process* part of the *Special Development Topics*.

Guidelines for Development

The guidelines you have to follow during development are quite simple:

- Always put *all* your application/adapter classes inside the *softwareag/appclasses* directory or in the corresponding project directories. When using the project management (which is strongly recommended), store the classes in the project directories so that you can easily copy projects as self-containing units between different Application Designer installations.
- Do *not* put classes into the servlet engine's class loader's class path.
- Avoid class duplicates (a *.class* file in the */classes* subdirectory also contained in a *jar* file inside the */lib* subdirectory).
- Reload the classes by creating a new class loader instance. To see the effects re-logon. (The re-logon can be done by refreshing the browser.)

Classpath Extensions in cisconfig.xml

In the *cisconfig.xml* file, you can define the possibility to explicitly include defined directories or *jar*/*zip*/etc. files in the application class loader. The following example shows a *cisconfig.xml* file containing a class loader extension:

```
<cisconfig ...>
    <classpathextension path="c:/development/centralclasses/classes/"/>
    <classpathextension path="c:/development/centralclasses/libs/central.jar"/>
</cisconfig>
```

Consequence: you can also include classes that are located outside the web application's directory structure into the application class loader of Application Designer.

Pay attention: if defining directories that contain *.class* files, then the path definition inside the classpath extension must end with a slash (/).

Loading Resource Files

The Application Designer application class loader does only load classes to be loaded into the Java virtual machine. It is not able to load resource files that you might access from your code.

Place resource files into the web application class loader, below the directory *<webapps>/WEB-INF/classes/* so that they are loaded in a correct way.

Preparing for Runtime

The following topics are covered below:

Basics

As explained in the previous section, the Application Designer class loader concepts are very useful for design time purposes. What is the price? The Application Designer class loader finds its classes by accessing the file system. It uses for this reason the `cis.home` parameter inside the *<webapp>/WEB-INF/web.xml* file in order to know the file root directory of the web application.

At runtime - especially if your application server distributes the load on several physical nodes - this is dangerous: each node may have its own directory structure and you cannot specify one root directory anymore in which the web application is located.

Consequence: for running in these scenarios, you have to prepare your application accordingly - i.e. you have to place your classes at the places where the application server definition defines them to be located.

The normal directories to put classes in are:

- *<webapp>/WEB-INF/lib* for libraries (*.jar* files).
- *<webapp>/WEB-INF/classes* for single class files (*.class* files).

In addition, you must switch off the flag "useownclassloader" inside the *cisconfig.xml*. Consequently, the Application Designer application class loader will not be used at all - all classes are loaded by the web application loader.

Example

Example: let us assume that you have set up the Application Designer application project "projectxyz". The classes for this project are located in

- *<webapp>/projectxyz/appclasses/classes/*.class* and
- *<webapp>/projectxyz/appclasses/lib/*.jar*

so that the Application Designer class loader can reach them.

For changing to the runtime scenario, just copy the *.*class* and *.*jar* files from your project directory into the corresponding standard directories.

6 Appendix E - StartCISPage Servlet

The StartCISPage servlet is the central servlet that is used in order to open intelligent HTML pages. It was already mentioned several times in this documentation. This chapter describes certain attributes that you can pass inside the servlet call.

Normal Calling of a Page

A normal page is called in the following way:

```
http://<host>:<port>/<webapplication>/servlet/StartCISPage?PAGEURL=/<project>/<pagename>.html
```

The StartCISPage servlet creates a frameset page around the intelligent HTML page that provides for specific functions that are internally required.

Appending Application Parameters

Application parameters can be passed by just appending the name and the value of the parameters to the URL. Each parameter must be the name of a property that is provided for by the server side adapter.

Example: the adapter provides for a property `company`. When opening a page via

```
http://<host>:<port>/<webapplication>/servlet/StartCISPage?PAGEURL=/<project>/<pagename>.html&company=softwareag
```

then the `setCompany` method of the adapter is called and the value "softwareag" is passed.

This is a very simple and powerful way to pass parameters through the URL.

Controlling the Session Life Cycle

A page relates to adapters living inside a session on server side. A session is opened by default when referencing a page via StartCISPage. By default, it is closed when the initial StartCISPage page is removed - either by closing the browser or by loading a different URL into it.

You can explicitly control this automated removal of sessions with the parameter `ONUNLOADBEHAVIOR`. If you call a page in the following way, the session is not removed when the page is removed:

```
http://<host>:<port>/<webapplication>/servlet/StartCISPage?PAGEURL=/<project>/<pagename>.html&ONUNLOADBEHAVIOUR=NOTHING
```

Controlling the Session ID

By default, a new session ID is internally generated when opening a page by StartCISPage. But you can also pass the session ID and the subsession ID explicitly. This might be of interest if you require to control the Application Designer session management from outside.

Calling a page in the following way

```
http://<host>:<port>/<webapplication>/servlet/StartCISPage?PAGEURL=/<project>/<pagename>.html&SESSIONID=4711&SUBSESSIONID=5
```

will internally open the session with ID 4711 - or use 4711 if it already exists. The same applies on subsession level.

Pay attention: if you use this possibility, then you are responsible for managing session IDs in such a way that they are unique.

Setting Default Parameters

Language

As described in *Multi Language Management*, Application Designer internally holds a language per session. This language can be set from outside:

```
http://<host>:<port>/<webapplication>/servlet/StartCISPage?PAGEURL=/<project>/<pagename>.html&LANGUAGE=E
```

Default Style Sheet

By calling

```
http://<host>:<port>/<webapplication>/servlet/StartCISPage?PAGEURL=/<project>/<pagename>.html&SESSIONCSS=../softwareag/styles/CIS_PARROTT.css&DEFAULTCSS=../softwareag/styles/CIS_PARROTT.css
```

you define that the CIS_PARROTT style sheet is used instead of the default style sheet. Of course, you can reference any style sheet of your own.

Mixing Parameters

All parameters can be mixed without any restrictions.

Setting Parameters with the HTTP Method POST

Instead of adding the parameters to the URL, you can also use the HTTP method `POST` to set the parameters in an HTML form.

Example (similar to the example under *Appending Application Parameters*, but with `POST`):

```
<html>
<head>
<title>Start Application Designer Demo Application</title>
<script type="text/javascript">
function submitStart() {
document.forms["myform"].submit();
}
</script>
</head>
<body>
 <form id="myform" name="myform" action="servlet/StartCISPage" method="post">
  <input type="hidden" name="PAGEURL" value="/<project>/<pagename>" />
   Company: <input type="input" name="company" value="softwareag" /><br/>
 </form>
 <a href="#" onclick="submitStart()">Start Demo</a>
 <div id="status">Click on Start Demo</div>
</body>
</html>
```