Uploading and Downloading Files in Web Dynpro Tables

Applies to:

Web Dynpro for Java UI Development, SAP NetWeaver 2004s

Prerequisite

This article is an addition to the tutorial *Uploading and Downloading Files in Web Dynpro Java in SAP NetWeaver 2004s* comprising more advanced technical details. Please read this tutorial first to understand the principles of file upload and download in SAP NetWeaver 2004s. Both articles are based on the same Web Dynpro sample application.

Summary

In SAP NetWeaver 04 file download and upload inside Web Dynpro *Table* UI elements was subject to significant technical restrictions which are all solved in the NetWeaver 04s release. This article and its related Web Dynpro sample project demonstrate how to upload and download image resources with different mime types per table line, how to store these image resources in a multiple context node of the controller context and how to instantly display an uploaded image in a table cell by utilizing the new Java dictionary type *Resource* and its related new Web Dynpro Java APIs. Additionally the article comprises an in-depth description of the new *on-demand streaming technique* combined with *0-byte resource creation* which yields a heavily reduced context memory consumption on server side.

Author: Bertram Ganz

Company: SAP AG

Created on: 19 February 2007

Author Bio

After his studies in mathematics, physics and computer science Bertram Ganz finished his teacher training at a German grammar school stressing technical sciences. He has been a member of the Web Dynpro Java Runtime development team (SAP NetWeaver ESI Foundation UI) since 2002. The main focus of his work is on knowledge transfer, rollout, and documentation. Bertram regularly publishes articles on Web Dynpro in the context of the SAP NetWeaver Application Server. Bertram is co-author of the books “Maximizing Web Dynpro for Java” and “Java Programming with the Web Application Server”.

Table of Contents

Introduction

In this article we discuss an interesting and important use case for the Web Dynpro UI elements *FileDownload* and *FileUpload*:

> *Uploading and downloading files in a Table UI element, or using the UI elements FileDownload and FileUpload as table cell editors.*

In contrast to the form-based examples in the first tutorial we now find answers on the following technical questions:

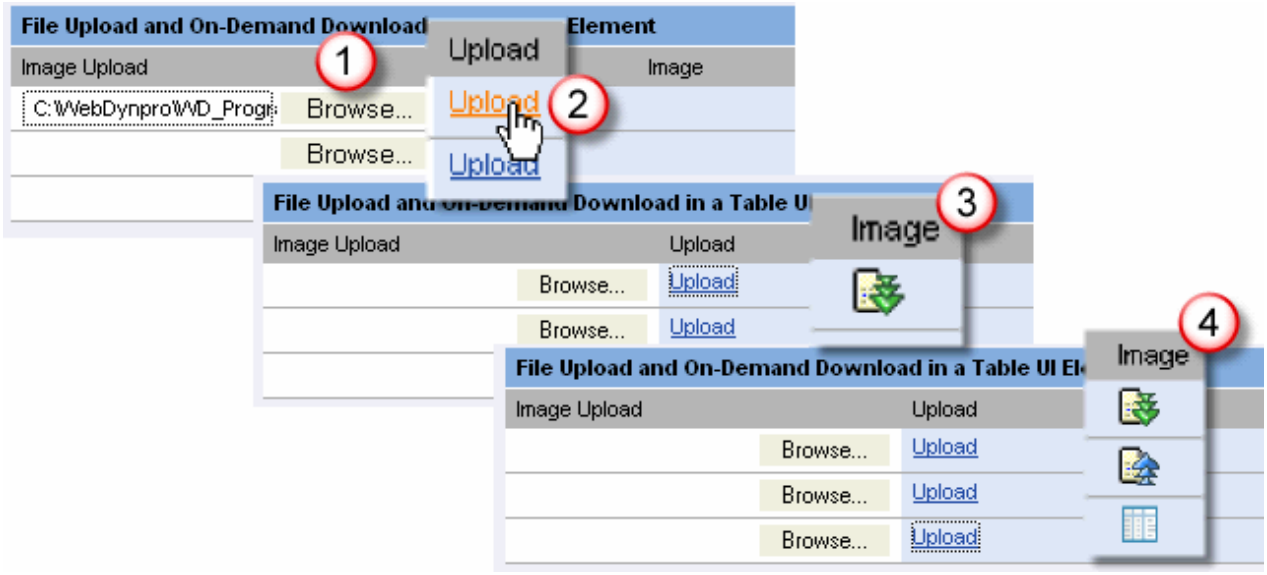
- How to upload image resources with different mime types (png, gif, jpg etc.) per table line?
- How to store these image resources in a multiple context node of the controller context?
- How to instantly display an uploaded image in a table cell?
- How to stream resource content (binary resource data) from server to client on-demand so that the controller so that it must not be initially streamed to the controller context befor the user actually downloads it?

What's New in SAP NetWeaver 04s

In SAP NetWeaver 04 file download and upload inside tables was subject to significant technical restrictions which are all solved in the NetWeaver 04s release. The solution is based on the following new features:

- **New Dictionary Type *Resource***: The new Java dictionary type *Resource* allows to bind FileUpload and FileDownload UI elements to context attributes storing MIME files named resources. In SAP NetWeaver 04 the primitive type binary must be used.
- **New Java Type *IWDResource***: The new Web Dynpro interface `IWDResource` is the Java type counterpart of the dictionary type *Resource*. Resource objects of this type can be created with the new Web Dynpro factory class `WDResourceFactory`.
- **Independant MIME Types**: In addition the new Java type `IWDResource` allows to store the file content (binary resource data) and the file metadata (MIME type, resource name) in one object. Consequently the MIME type can differ among the context node elements stored in the table's data node. In SAP NetWeaver 04 the storage of file content and file metadata was separated between context attribute info (`IWDAttributeInfo`) on context node level and context attribute (byte array) on context nodel element level. Consequently it was not possible to upload or download files with different MIME types in a *Table* UI element.
- **Downloading Files in Tables On-Demand**: The new *on-demand streaming technique* allows to download the resource content *on-demand* when the user actually requests it on client side. Especially when using the *FileDownload* UI element as table cell editor this new technique yields a heavily reduced context memory consumption on server side. Initially, the controller context stores so-called *0-byte resources* which do not comprise any binary content yet but only the resource metadata like MIME type and file name. Every requested binary resource content gets streamed to the client *on-demand* by invoking a calculated context attribute getter method of a Web Dynpro controller on server side.

Uploading Files in a Table



Within this section we demonstrate how to realize a simple use case of a *FileUpload* table cell editor (see screenshot above):

1. The user selects an image file in the client's file system.
2. The image resource is uploaded to the server by clicking a *LinkToAction* UI element in the next table column.
3. The uploaded image is instantly displayed in another table column (*Image*).
4. The mime type of the uploaded images can be different in all table rows.

Context Definition and Data Binding

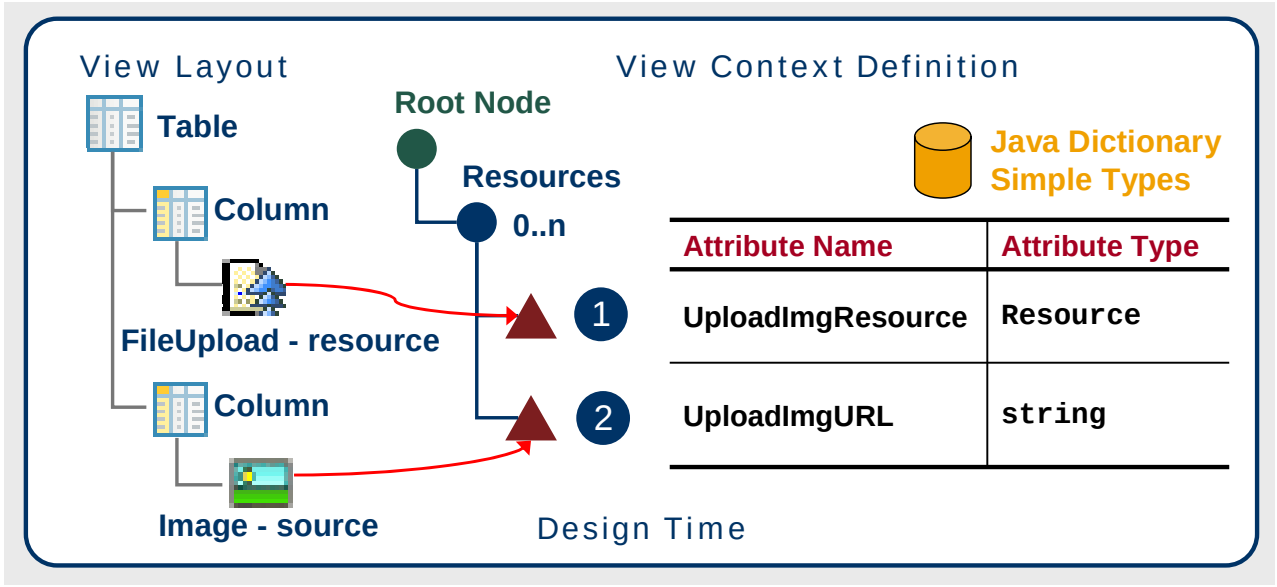


Figure 1: Defining Data Binding and Context for File Upload in a Table at Design time

The context and data binding definition for our *file-upload-in-table scenario* is quite simple. The property *resource* of the *FileUpload* table cell editor is bound to the context attribute *UploadImgResource* of dictionary type *com.sap.ide.webdynpro.uielement-definitions.Resource* (see figure 1, point 1). After uploading the image file to the server the context attribute in the related node element stores it as an object of type *Resource*.

To instantly display this image resource in an *Image* table cell editor, we bind its *source* property to the context attribute *UploadImgURL* of type *String* (2). This means we must programmatically get the URL of the uploaded resource object which is stored in context attribute *Resource.* We will see later, that this URL can easily be retrieved by invoking the `IWDResource`-API.

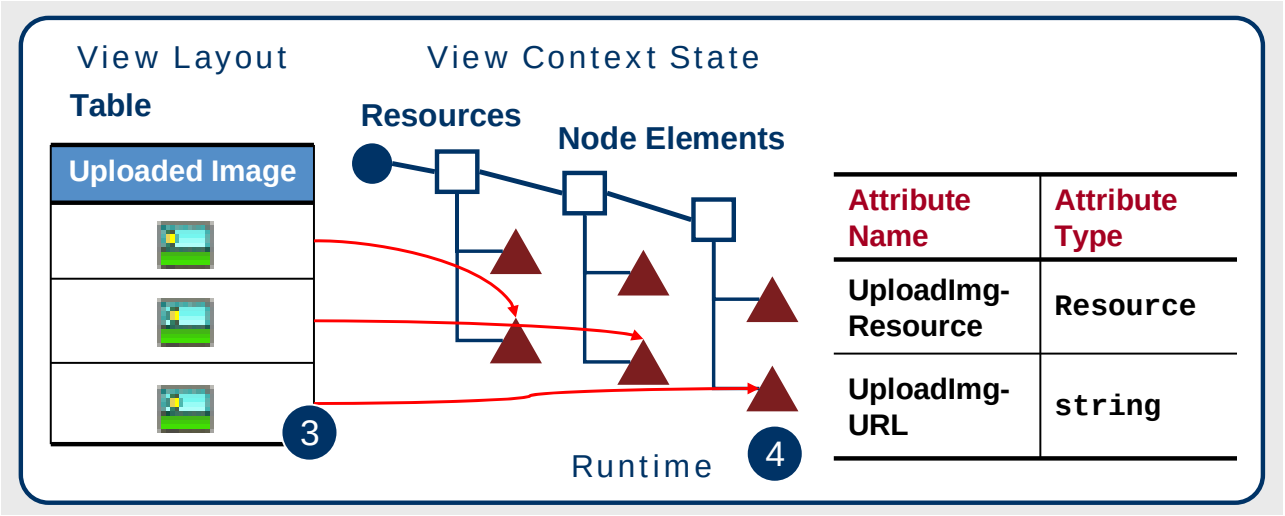


Figure 2: Data binding of Uploaded Images to the Context at Runtime

At runtime all *Image* UI elements in table column *Image* (see figure 2, point 3) are bound to the context attribute *UploadImgURL* of the corresponding context node element (4). As the MIME type information is associated with the resource objects on node element level but not with the context attribute information (of type `IWDAttributeInfo`) on node level all uploaded images can have different MIME types.

Event Parameter Based Table Interaction

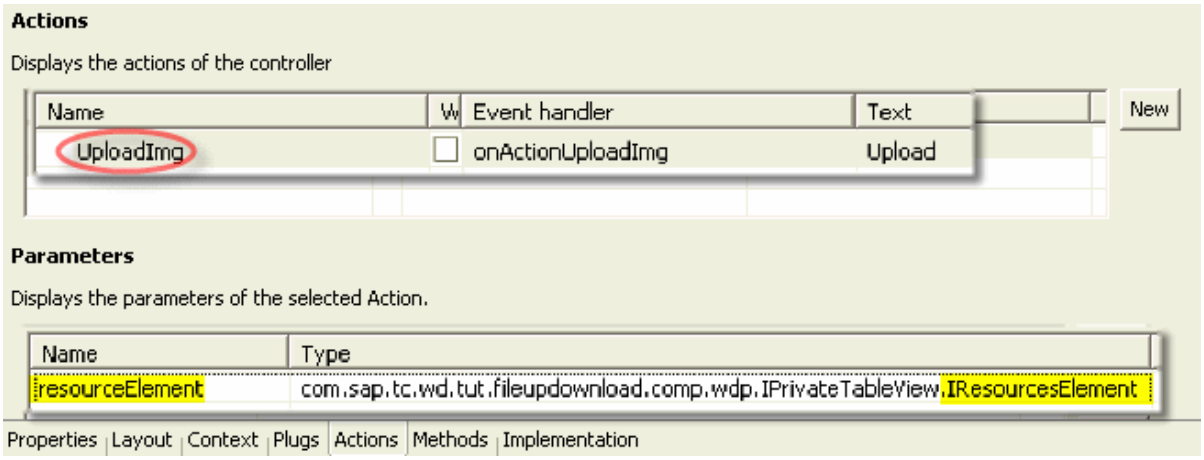
When the user clicks the *LinkToAction* UI element the selected image resource is uploaded to the server. Within the related action event handler we must know the table line in which the user selected a local image file so that we can get the required image URL.

Technically speaking we must get a reference to either the table's lead selection or to the node element directly. As we want to avoid an implicit lead selection change when the user selects a table line we apply another table interaction technique: **parameter mapping** in combination with the table *compatibiltyMode nw04plus* (which is identical with the property value *auto* for all components newly created in NW04s). With this approach a reference to the action-triggering node element is automatically passed to the corresponding action event handler by the Web Dynpro Java Runtime.

Defining Action Uploading

Within the *TableView* controller we define a new Action:

- **Name**: *UploadImage*
- **Text** : *Upload*. This text will be displayed in every line of table column *Upload*.
- **Parameter**: *resourceElement* of type *IPrivateTableView.IResourcesElement*. This is the generated context interface for the node element objects in the data node *Resources* to which the *Table* UI element is bound.



Implementing the Parameter Mapping Relation

The parameter mapping relation between the *LinkToAction* UI element event parameter `nodeElement` and the action parameter `resourceElement` is implemented within the `wdDoModifyView()` hook method:

Parameter Mapping – TableView.java

```
public static void wdDoModifyView(IPrivateTableView wdThis,
  IPrivateTableView.IContextNode wdContext,
  com.sap.tc.webdynpro.progmodel.api.IWDView view, boolean firstTime)
{
  //@@begin wdDoModifyView
  if (firstTime) {
    IWDLinkToAction linkToAction =
      (IWDLinkToAction) view.getElement("UploadImgLinkToAction");
    linkToAction.mappingOfOnAction()
      .addSourceMapping("nodeElement", "resourceElement");
  }
  //@@end
}
```

Implementing the Upload Action Event Handler

Within the action event handler `onActionUploadImg()` we finally get the URL of the uploaded image resource of type `IWDResource` and copy it to the context attribute *UploadImgURL* so that the image is instantly displayed on the UI.

The URL of an object of type `IWDResource` can easily be retrieved by invoking the interface method `IWDResource.getURL(int fileDownloadBehavior)`. The integer value for the parameter `fileDownloadBehavior` can be calculated with the method `WDFileDownloadBehavior.ordinal()`. To display the uploaded image file *in-place* or *in-table* we pass the ordinal integer value for the constant `WDFileDownloadBehaviour.OPEN_INPLACE`.

Upload Action Event Handler – TableView.java

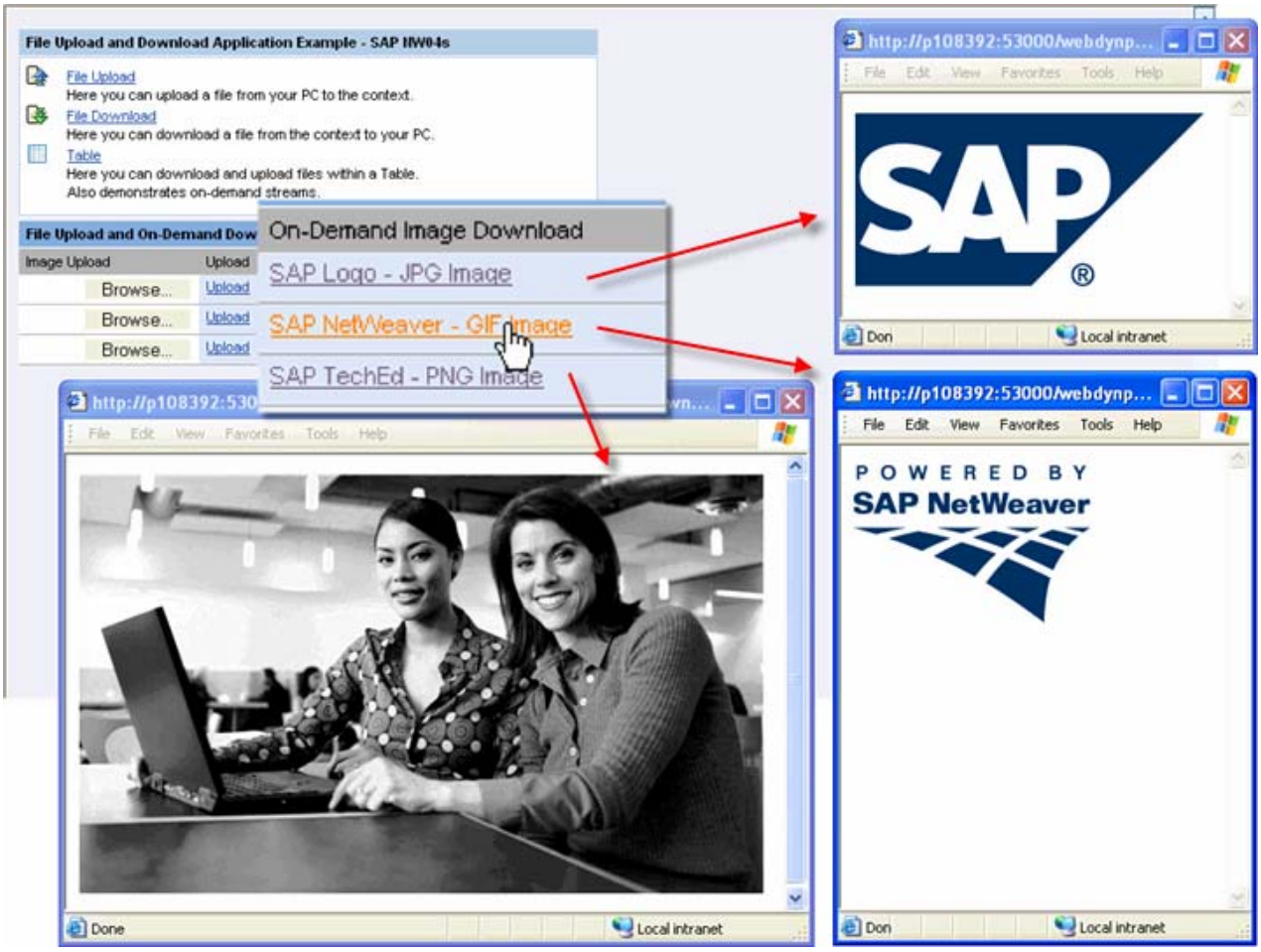
```
//@@begin javadoc:onActionUploadImg(ServerEvent)
/** Declared validating event handler. */
//@@end
public void onActionUploadImg(
  com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent wdEvent,
  com.sap.tc.wd.tut.fileupdownload.comp.wdp
    .IPrivateTableView.IResourcesElement resourceElement )
{
  //@@begin onActionUploadImg(ServerEvent)
```



```
  //  if a file in the FileUpload field exists
  if (resourceElement.getUploadImgResource() != null) {
    // copy URL string from IWDResource object to
    // context attribute 'UploadImgURL'. Image gets
    // visible on the UI based on data binding definition.
    resourceElement.setUploadImgURL(
      resourceElement.getUploadImgResource().getUrl(
        WDFileDownloadBehaviour.OPEN_INPLACE.ordinal()));
  }
  //@@end
}
```

Within SAP NetWeaver 04 we must use the *Web Dynpro Binary Cache* service to create an object of type `IWDCachedWebResource` from the image byte array stored in the context.

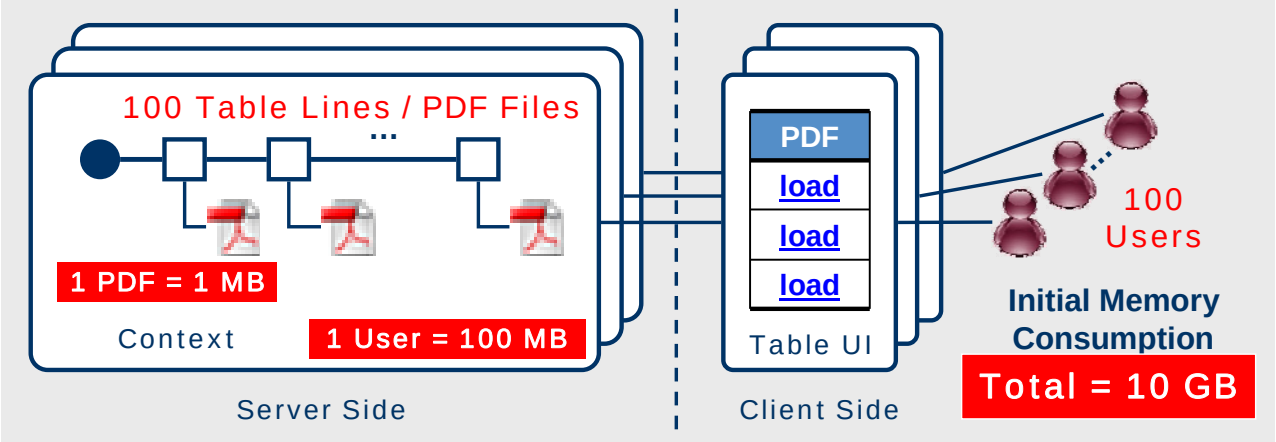
Downloading Files in a Table Using On-Demand Streams



When using a *FileDownload* UI element as table cell editor in NW 04 we must be aware of the potentially high memory consumption within the controller context on server side. To understand this problem let's look at the following file download example.

Example

Please note, that the technical description of this example is based on SAP NetWeaver 04. Afterwards we will present a solution which can only be implemented in SAP NetWeaver 04s using *on-demand streams*.



A *FileDownload* UI element is used as cell editor inside a *Table* UI element in order to download a contract PDF document for every displayed customer. We assume that every PDF document has 1MB file size and that the table displays 100 customers.

To download one single PDF file on client side we must initially store the content of all 100 PDF files as byte arrays within the controller context, in total **100 MB**. Although Web Dynpro supports table paging on client side (only the data of the visible rows is sent to the client) it does not support context paging on server side. This means that a supply function initially populates the data node with all 100 node elements which can be potentially displayed in the table. Every node element stores the byte array of 1 PDF resource in a context attribute of type *binary*.

Now we assume that our customer table view is simultaneously displayed by 100 concurrent users. As every user is running in its own Web Dynpro session the total memory consumption for all PDF files in the controller contexts on server side sums up to **10 GB**.

The high memory consumption is based on the fact, that all downloadable resources must be fully stored in the controller context's node elements **in advance** before the user actually requests or downloads them on the client. This is based on the following circumstances:

- **Data Binding**: The *FielDownload* UI element property *data* must be bound to the context attribute storing the binary data (content) of the downloadable resource. In NW 04 the Web Dynpro Runtime stores this resource in its binary cache and sends the related download URL to the client.
- **UI Element Event Model**: The *FielDownload* UI element does not provide an action event like *onDownload* which triggers a roundtrip and which could be handled in an action event handler on server side. Consequently the application developer has no possibility to initialize a binary context attribute *lazily* before the user actually downloads it and to store the corresponding byte array in it on-demand.

Solution in SAP NetWeaver 04s: Using On-Demand Streams

In SAP NetWeaver 04s we can apply a new function in Web Dynpro which solves the above memory consumption problem:

- **Context Attribute of type *Resource***: The *FileDownload* UI element must be bound to a context attribute of type *com.sap.ide.webdynpro.uielementdefinitons.Resource*.
- **0-Byte Resource Creation**: At runtime we initially create so-called *0-byte resource objects* and store them in the controller context.
  A *0-byte resource* is an object instance of type `IWDResource` comprising resource *metadata* but no binary resource *content* (0-byte). This means we just set the variables *resourceName* (file name) and *resourceType* (MIME type) and then store these resource objects in the context initially. The resource content (byte arrays, `java.io.InputStream` or `IWDInputStream` objects) is streamed to these resource objects later on-demand in case the client requests it (*on-demand streaming*). How to create a 0-byte resource object of type `IWDResource` with the `WDResourceFactory` class will be described in section Populating the Context Node Resources and Creating 0-Byte Resources.

- **Calculated Context Attribute and On-Demand Streaming**: Instead of streaming the downloadable binary data to the context initially it is streamed to the client on-demand. This is achieved by invoking the getter method of an additional *calculated context attribute* which is referenced by the 0-byte resource object and which returns an object of type `IWDInputStream`. The required server roundtrip is automatically triggered when the user clicks the file download link in the Web Dynpro view layout.

Summary

The process of *0-byte resource creation* doesn't initially allocate memory for the file content of the resource object but only for the file metadata. It enables the streaming of this content to be put off until it is actually requested by the client. Instead of being streamed to the context when the table view first gets visible on the UI, the streaming of the file content is deferred until the client actually downloads it and until the calculated context attribute getter method is invoked in the controller on server side. Initially the client only receives the *metadata* (URL, file name, MIME type) which is needed to prepare the later download of the file *content* (in case it is triggered by the user).

Context Definition and Data Binding

The following diagrams illustrate the context definition and data binding relation of the on-demand stream file download technique:

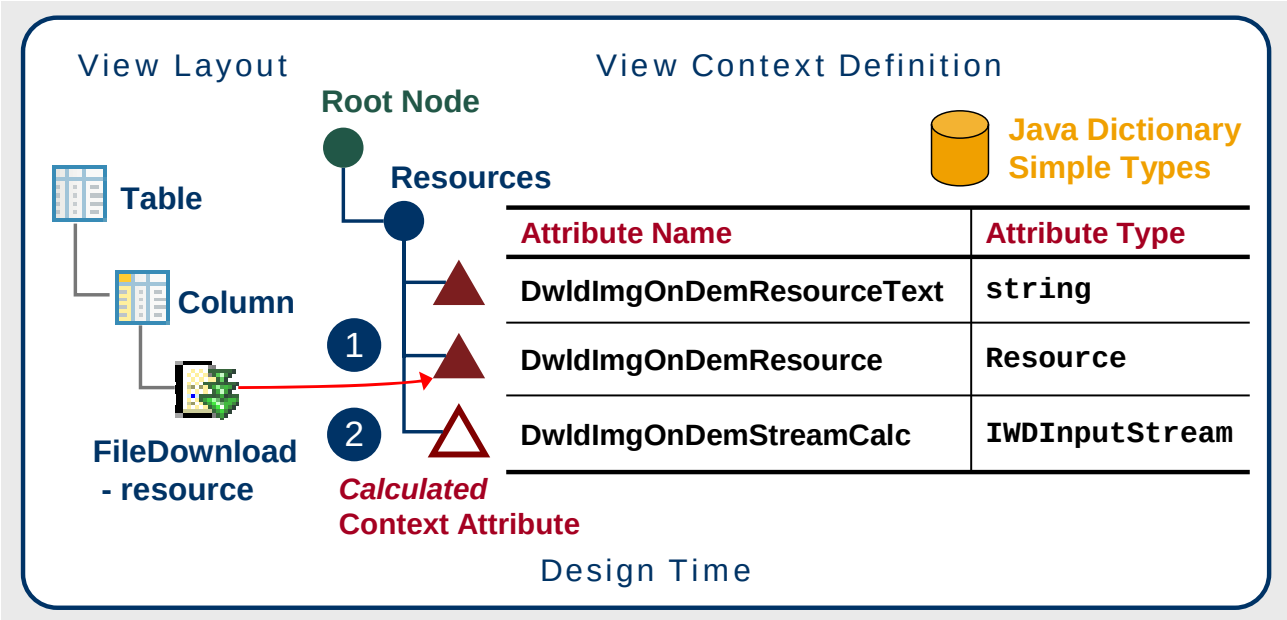


Figure 3: Data binding and Context Definition for On-Demand File Download at Design time

5. The *FileDownload* UI element property *resource* is bound to the context attribute *DwldImgOnDemResource* of dictionary type *Resource* (see figure 3, point 1).
6. Additionally a read-only *calculated* context attribute *DwldImgOnDemStreamCalc* of type `IWDInputStream` must be defined (2). There is no data binding relation between the *FileDownload* UI element and this calculated context attribute. The relation between the context attribute *DwldImgOnDemResource* of type `IWDResource` to the calculated context attribute *DwldImgOnDemStreamCalc* of type `IWDInputStream` is implemented in the controller code.

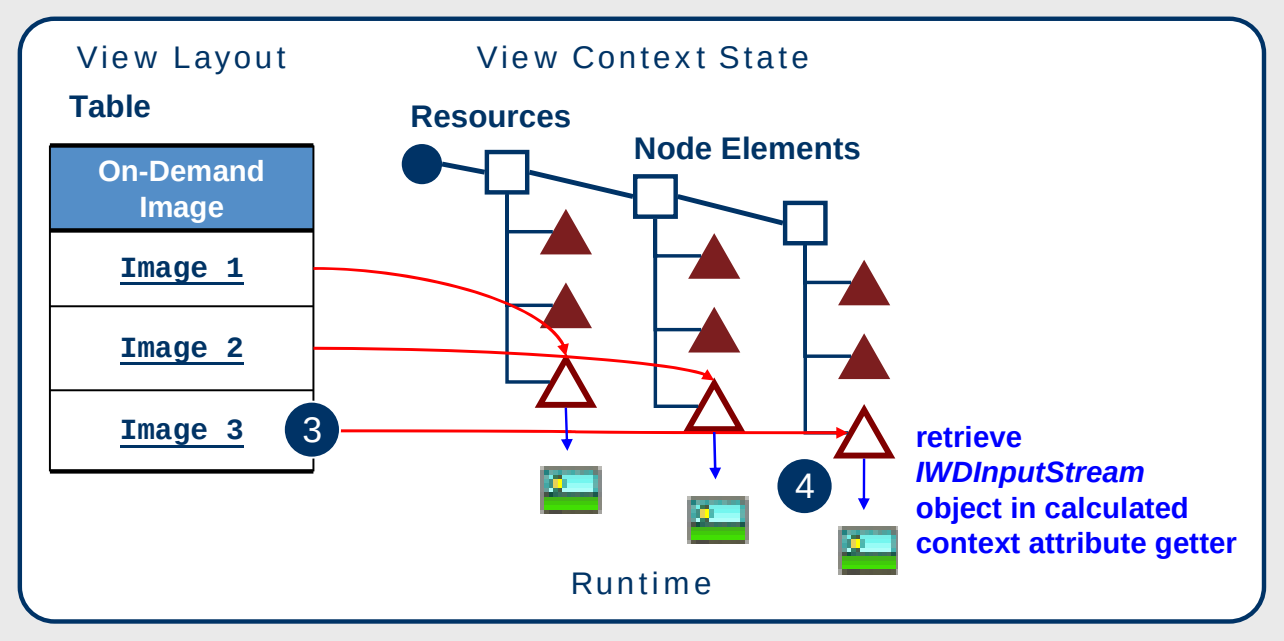


Figure 4: On-Demand File Download in a Table at Runtime

When the user triggers a file download on client side (figure 4, 3) the Web Dynpro Runtime invokes the getter method of the calculated context attribute *DwldImgOnDemStreamCalc* returning the file content or the binary resource data to be streamed to the client (figure 4, 4).

Populating the Context Node Resources and Creating 0-Byte Resources

Now let's have a look at the controller code for our on-demand stream solution. To stream files from the file or backend system on server side to the client on-demand we must instantiate *empty* instances of type `IWDResource` when populating the context data node. A *data node* is a the context node to which the *Table* UI element is bound.

In this sample application we just populate the data node with three node elements so that the table comprises only three table lines (implemented in supply function `supplyResources()`.

0-Byte Resource Creation

Every node element stores an image resource of another MIME type to be downloaded by the client. The creation of a *0-byte resource* object is implemented by invoking a special method of the `WDResourceFactory` class:

```
WDResourceFactory.createResource(
  IWDAttributePointer attributePointer,
  java.lang.String resourceName,
  WDWebResourceType type)
```

We **do not** pass any resource *content* as a byte array to the resource factory but we only pass a pointer to the calculated context attribute instance in the same node element instead.

To avoid code redundancy we encapsulate the creation of a 0-byte resource in the private controller method `create0ByteResource()`.

Supply Function – TableView.java

```
public void supplyResources(IPrivateTableView.IResourcesNode node,
  IPrivateTableView.IContextElement parentElement) {
  //@@begin supplyResources(IWDNode,IWDNodeElement)
  String calcAttrName =
    IPrivateTableView.IResourcesElement.DWLD_IMG_ON_DEM_STREAM_CALC;
  IPrivateTableView.IResourcesElement resourceElement;
    // ----------- 1. Resource Node Element ------------------------------
    resourceElement = wdContext.nodeResources().createResourcesElement();
    node.addElement(resourceElement);
    resourceElement.setDwldImgOnDemResourceText("SAP Logo - JPG Image");
    resourceElement.setDwldImgOnDemResource(
      this.create0ByteResource (
        resourceElement,
        calcAttrName,
        "SAPLogo.jpg",
        WDWebResourceType.JPG_IMAGE));
    // ----------- 2. Resource Node Element ------------------------------
    resourceElement = wdContext.nodeResources().createResourcesElement();
    node.addElement(resourceElement);
    resourceElement.
      setDwldImgOnDemResourceText("SAP NetWeaver - GIF Image");
    resourceElement.setDwldImgOnDemResource(
      this.create0ByteResource (
        resourceElement,
        calcAttrName,
        "SAPNetWeaver.gif",
        WDWebResourceType.PNG));
    // ----------- 3. Resource Node Element ------------------------------
    resourceElement = wdContext.nodeResources().createResourcesElement();
    node.addElement(resourceElement);
    resourceElement.setDwldImgOnDemResourceText("SAP TechEd - PNG Image");
    resourceElement.setDwldImgOnDemResource(
      this.create0ByteResource (
        resourceElement,
        calcAttrName,
        "SAPTechEd.png",
        WDWebResourceType.GIF_IMAGE));
  //@@end
}
...

//@@begin others
/**
 * Returns a 0-byte resource object of type IWDResource which points to a
 * calclated attribute instance.
 * The getter method of this calculated attribute is invoked by the Web
 * Dynpro Java Runtime after the user triggered the file download on client
 * side and streams the file content back to the client on-demand.
 */
private IWDResource create0ByteResource(
  IWDNodeElement dataNodeElement,
  String calcAttrName,
  String resourceName,
  WDWebResourceType resourceType) {
  return WDResourceFactory.createResource(
    dataNodeElement.getAttributePointer(calcAttrName),
    resourceName,
    resourceType);
}
//@@end
```


Implementing the Calculated Context Attribute Getter Method

After having implemented the supply function for the table's data node *Resources* and after having created 0-byte resource objects of type `IWDResource`, we now implement the getter method of the calculated context attribute of type `IWDInputStream`. In this method we finally stream the binary content of the requested file or resource from the server to the Web Dynpro client *on-demand*.

Calculated Attribute Getter Method – TableView.java

```
//@@begin
javadoc:getResourcesDwldImgOnDemStreamCalc(IPrivateTableView.IResourcesElement)
/**
 *  Getter method for calculated ttribute OnDemandStreamCalc of node Resources.
 *  Return IWDInputStream object for given on-demand resource, which is deployed
 *  in the same deployable object part (Web Dynpro Component 'FileUpDownloadComp').
 *  @param element the element requested for the value
 *  @return the calculated value for attribute DwldImgOnDemStreamCalc
 */
//@@end
public com.sap.tc.webdynpro.progmodel.api.IWDInputStream
  getResourcesDwldImgOnDemStreamCalc(
    IPrivateTableView.IResourcesElement element) {
    //@@begin getResourcesDwldImgOnDemStreamCalc(IPrivateTableView.IResourcesElement)
  IWDInputStream onDemandStream = null;
  try {
    // ===== SIMPLE CODE only running in SAP NW 04s Stack 12 ========
    // Based on a bug this simple coding cannot be implemented. The bug
    // will be fixed within NW04s SP Stack 12.
    /*IWDResource resource =
        WDWebResource.getWebResource(
          wdComponentAPI.getDeployableObjectPart(),
          element.getDwldImgOnDemResource().getResourceType(),
          element.getDwldImgOnDemResource().getResourceName());
    InputStream inputStream = resource.read(false);
    return WDResourceFactory.createInputStream(inputStream);*/

    // ===== WORKAROUND CODE ====================================

    // Get path of the deployed web resource using the WDURLGenerator service.
    // NOTE: In a real life application the input stream is retrieved by executing a
    // corresponding model object (Adaptive RFC, Web Service).
    String resourcePath =
      WDURLGenerator.getResourcePath(
        wdComponentAPI.getDeployableObjectPart(),
        element.getDwldImgOnDemResource().getResourceName());

    // Create file input stream for resource path.
    // NOTE: The file input stream cannot be closed here. As it is created on-demand
    // this is a minor restriction.
    FileInputStream fileInputStream = new FileInputStream(new File(resourcePath));

    // Create an object of type IWDInputStream by passing the file input stream to the
    // WDResourceFactory service.
    onDemandStream = WDResourceFactory.createInputStream(fileInputStream);
  } catch (IOException e) {
    wdComponentAPI.getMessageManager().reportMessage(
      IMessageFileUpDownloadComp.FNF,
      new Object[] { element.getDwldImgOnDemResource().getResourceName()},
      true);
  } catch (WDAliasResolvingException e) {
    wdComponentAPI.getMessageManager()
      .reportException(e.getLocalizedMessage(), true);
  }
  return onDemandStream;
```



```
  //@@end
}
```

Within the calculated attribute getter method `getResourcesDwldImgOnDemStreamCalc()` we create an object of type `IWDInputStream` by passing a file input stream to the `WDResourceFactory` service class.

In this way the *0-byte resource creation* implemented in the supply function gets finally completed *on-demand* by adding the missing content of the resource object which is already stored in the context attribute *DwldImgOnDemResource.*

Keep in mind, that the streamed resource will not be re-calculated again after the first invocation of the calculated context attribute getter method, in our case `getResourcesDwldImgOnDemStreamCalc()`. After the first invocation of the getter method, the resource is streamed to the client and keeps cached in the Web Dynpro Binary Cache. The Web Dynpro Runtime does not re-invoke the calculated context attribute getter method for a completely initialized, cached resource which is already streamed to the client. Only in case the context attribute stores a *0-byte resource* comprising no content the calculated context attribute getter method gets invoked.

Note

In this example all image resources are deployed with the tutorial component, stored under the folder `src/mimes/Components/com.sap.tc.wd.tut.fileupdownload.comp.FileUpDownloadComp`, so that they can be easily retrieved using the Web Dynpro services `WDURLGenerator` and `WDResourceFactory`. In a real Web Dynpro application scenario the file content must be retrieved by executing a corresponding model object (Adpative RFC, Adaptive Web Service) in the calculated context attribute getter method.

Related Content

SDN Tutorial: Uploading and Downloading Files in Web Dynpro Java – SAP NetWeaver 04s

SAP Online Help: File Upload and File Download

Sample Application

This article is based on a Web Dynpro sample application, which is available for download on the SAP Developer Network (SDN) http://sdn.sap.com under

1. *SDN Home* → Developer Areas → Web Application Server → Web Dynpro → How-To Guides → Web Dynpro Sample Applications and Tutorials
2. Uploading and Downloading Files in SAP NetWeaver 04s (39)
3. Code Sample: Web Dynpro project *TutWD_FileUpDownload_NW04s*

Prerequisites

The sample application is based on a SAP NetWeaver 04s – SP Stack 8 installation.

Copyright