

How to Upload and Download Files in a Web Dynpro for Java Application



Applies to:

Web Dynpro for Java 7.11. For more information, visit the [Web Dynpro Java homepage](#).

Summary

In this tutorial you learn how to download and upload files within Web Dynpro applications by utilizing the dictionary type *Resource* and its related Web Dynpro APIs `IWDResource` and `WDResourceFactory` which are available since SAP NetWeaver 04s. In contrast to the dynamic type modification of a binary context attribute in SAP NetWeaver 04, the dictionary simple type *Resource* yields a fully declarative, zero coding data transport of download and upload resources between Web Dynpro client and controller context on server side.

Author: Web Dynpro Java Team

Company: SAP AG

Created on: 29 June 2010

Table of Contents

Prerequisites.....	3
Objectives.....	3
Some Theory.....	4
Using the Virus Scan Interface.....	4
The Tutorial Application.....	5
Implementation Details.....	8
Uploading Files.....	8
Downloading Files.....	10
Extended Upload and Download features.....	12
Uploading Files in a Table.....	12
Downloading Files in a Table Using On-Demand Resources.....	15
Solution 1: Using On-Demand Streams and Calculated Context Attribute	16
Further Information.....	25
Legend.....	25
Text Symbols.....	25
Copyright.....	26

Prerequisites

You need to install the NetWeaver Developer Studio (Version 7.11 or later) in order to compile and deploy the tutorial application. The SAP Java AS to which this application is deployed should have the same or newer version as the NWDS.

The tutorial application is available as a development component (DC). You need to import the software component HM-WDUIDMKT CNT, which contains the DC `tc/wd/tut/file/updownld`. The exact steps are described in a separate document.

Objectives

By the end of this tutorial, you will be able to:

- Use the UI element *FileUpload*.
- Use the UI element *FileDownload* with different file download behaviors
- Assign the dictionary simple type *Resource* to context attributes so that they can store MIME files.
- Create objects of type `IWDResource` from an image resource deployed with the project by invoking the `WDResourceFactory` API.
- Implement On-Demand Resources for Downloads.

Some Theory

The Web Dynpro UI Element Library provides two special UI elements (*FileDownload* and *FileUpload*) with which you can download files from the Web Dynpro runtime environment or upload them there. This is performed using declarative *data binding*. Here, Web Dynpro runtime automatically transports different types of MIME files between the client-side user interface or UI element and the server-side controller context.

In contrast to the semi-declarative approach in SAP NetWeaver 04, which was based on invoking the `IWDModifiableBinaryType`-API in the controller code, the new dictionary simple type *Resource* within SAP NetWeaver 04s yields a fully-declarative data transport of MIME resources between Web Dynpro client and controller context on server side. The old `IWDModifiableBinaryType`-API should not be used anymore since SAP NetWeaver 04s. The reasons for this are described in the chapter *Implementation Details*.


In SAP NetWeaver 04s file download and upload was significantly enhanced and simplified with the following new features:

- **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.
- **Java Type *IWDResource*:** The new Web Dynpro interface `IWDResource` is the Java type counterpart of the dictionary type *Resource*. It allows to store the file content (binary resource data) and the file metadata (MIME type, resource name) in one object. Consequently the resource metadata (MIME type, file name) must no longer be stored in the context attribute info (`IWDAttributeInfo`) using a modifiable binary type and may therefore differ among multiple resources or node elements stored in the same context node.
- **Web Dynpro Factory *WDRResourceFactory*:** Resource objects of type `IWDResource` can easily be created with the new Web Dynpro factory class `WDRResourceFactory`. This factory class significantly simplifies the implementation of file download scenarios where statically deployed or dynamically created resources must be stored in the context.
- **Zero Coding File Upload:** With the new Java dictionary type *Resource* it is no longer needed to implement the type modification of a binary context attribute by invoking the `IWDModifiableBinaryType` API in the controller code. Consequently file upload can be realized in a purely declarative, zero coding approach. You just have to bind a *FileUpload* UI element to a context attribute of type *Resource*. The Web Dynpro Runtime then automatically transports the uploaded file to the context attribute as an object of type `IWDResource`.
- **File Download Behaviors:** The behavior of the *FileDownload* UI Element can now be defined with the new Java dictionary type *FileDownloadBehavior*. Its enumeration specifies three different file download behaviors: open resource in-place without opening a dialog window, save resource in local file system (open dialog) and open resource depending on the MIME type of the downloaded file (open dialog).
- **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.

Using the Virus Scan Interface

To enhance your system's virus protection when working with files or documents processed by your Web Dynpro applications, you can add external virus scanners to your SAP system using the Virus Scan Interface.

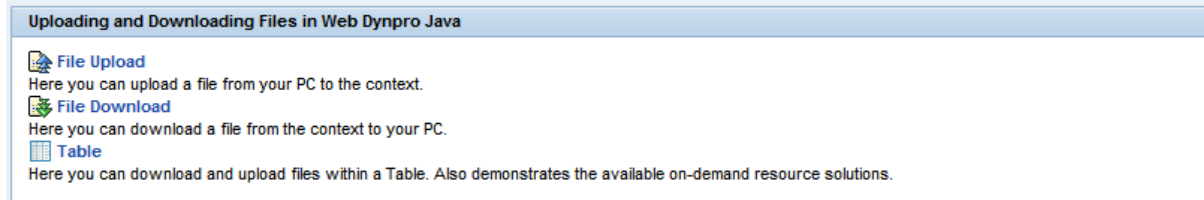
To connect the *FileUpload* service contained in the Web Dynpro runtime environment to a virus scanner, you need to activate the predefined virus scan profile `webdynpro_FileUpload`. This profile must be activated/deactivated by the SAP J2EE Engine administrator. When delivered, profile `webdynpro_FileUpload` is switched off. The Virus Scan Interface cannot be used for the Web Dynpro *FileDownload* service.

 Note that the example application presented in this tutorial does not use the Virus Scan Interface. More details can be found in the SAP NetWeaver Help (see the link “Delivered Virus Scan Profiles”) in the chapter *Further Information*.

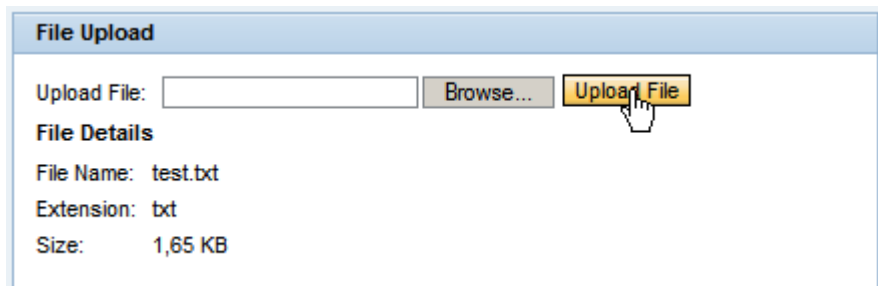
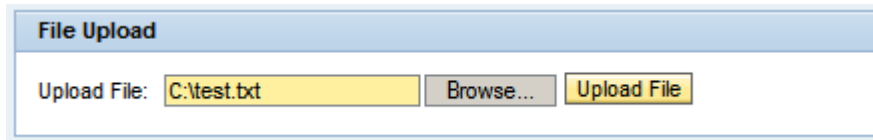
The Tutorial Application

The screenshots displayed below show the views in the tutorial application.

When the application is launched, the *WelcomeView* appears, where you can navigate to the different scenarios *File Upload*, *File Download* and *Table*.

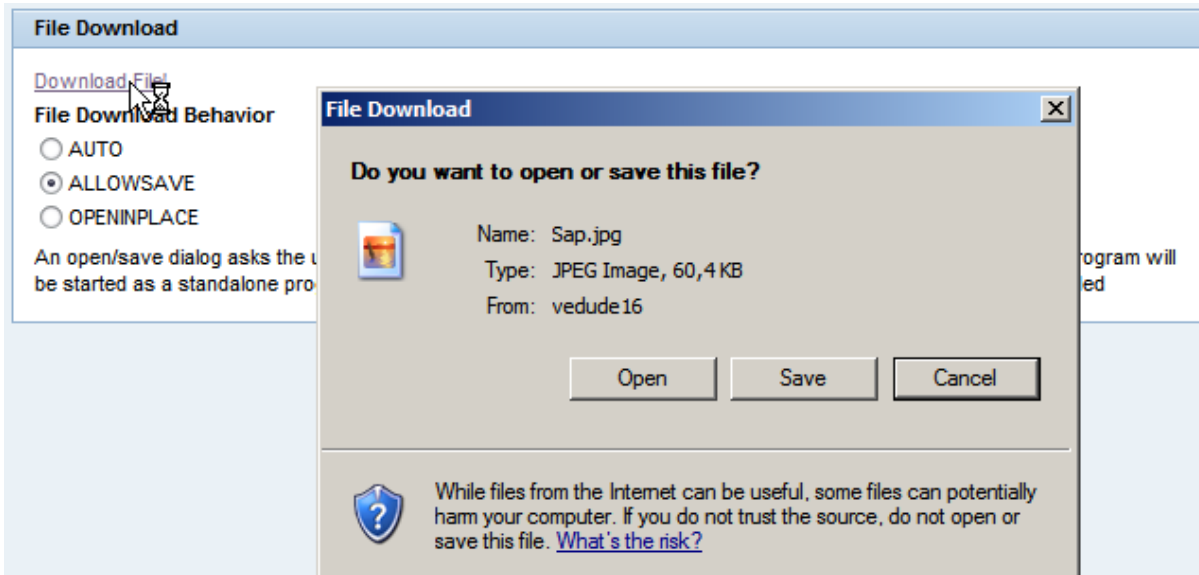


When you press the Link “File Upload” the *FileUploadView* becomes visible. You can upload a file from your computer to the server-side controller context. Once a MIME object has been uploaded, the system displays details of the uploaded file, such as file name, file ending and file size.



In the *FileDownloadView* (reachable via the link “File Download”), you can download an image file, which is deployed in the example project, from the controller context and display it on the user interface.

The file download behavior can be defined by clicking one of three radio buttons. The following screenshot shows that the file download behavior `ALLOWSAVE` opens a dialog window to save the downloaded resource on the local file system. With the behavior `OPENINPLACE` the resource is instantly displayed in another browser window.



Via the link “Table” you can show the *TableView* which shows how to use the UI elements *FileDownload* and *FileUpload* as table cell editors. In contrast to the form-based examples in the first two screens this view deals in addition with the following questions (Find more implantation details in the chapters below):

- 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 it must not be initially streamed to the controller context before the user actually downloads it?

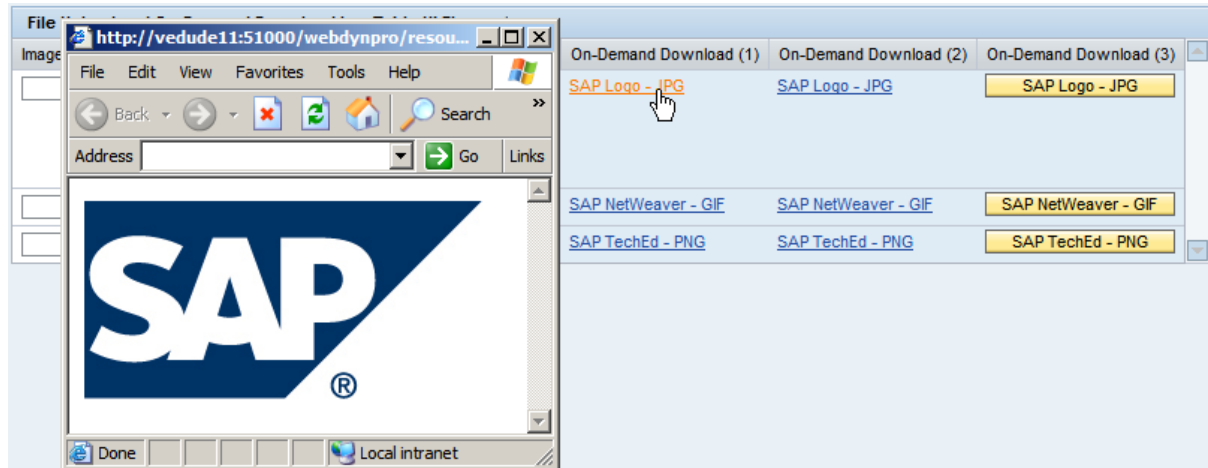
The initial *TableView* looks like follows.

File Upload and On-Demand Download in a Table UI Element					
Image Upload	Upload	Image	On-Demand Download (1)	On-Demand Download (2)	On-Demand Download (3)
<input type="text"/> Browse...	Upload		SAP Logo - JPG	SAP Logo - JPG	SAP Logo - JPG
<input type="text"/> Browse...	Upload		SAP NetWeaver - GIF	SAP NetWeaver - GIF	SAP NetWeaver - GIF
<input type="text"/> Browse...	Upload		SAP TechEd - PNG	SAP TechEd - PNG	SAP TechEd - PNG

By using the *FileUpload* UI elements and pressing the link “Upload” you can upload an image and display it in the corresponding column “Image”.

File Upload and On-Demand Download in a Table UI Element					
Image Upload	Upload	Image	On-Demand Download (1)	On-Demand Download (2)	On-Demand Download (3)
<input type="text"/> Browse...	Upload		SAP Logo - JPG	SAP Logo - JPG	SAP Logo - JPG
<input type="text"/> Browse...	Upload		SAP NetWeaver - GIF	SAP NetWeaver - GIF	SAP NetWeaver - GIF
<input type="text"/> Browse...	Upload		SAP TechEd - PNG	SAP TechEd - PNG	SAP TechEd - PNG

Via the links and buttons in the columns “On-Demand Download (X)” you can download predefined images with different mime types. Each column uses another technique for the On-Demand Resource creation.



Implementation Details

Uploading Files

This section describes the implementation details of the *FileUploadView*.

To save a MIME file in the controller context after it has been uploaded from the user interface to the server, you first need to define a context attribute of type *com.sap.ide.webdynpro.uelementdefinitions.Resource*.

The *Resource* type is a special dictionary simple type for MIME resources. At runtime the controller context attribute stores the MIME resource in an object of type

com.sap.tc.webdynpro.services.sal.datatransport.api.IWDResource.

Property	Value
Calculation	Calculated
Calculated	false
Misc	
Name	fileResource
Read-Only	false
Semantic ID	
Structure Element	
Type	com.sap.ide.webdynpro.uelementdefinitions.Resource

After defining the context attribute *FileUpload* of type *Resource*, you can bind a new *FileUpload* UI element to it. To trigger uploading of a selected file on the user interface, a *Button* UI element (*UploadButton*) is inserted next to it and its *onAction* event is bound to the *UploadFile* action.

Property	Value
Properties[FileUpload]	
id	FileUpload
layoutData	MatrixData
activateAccessKey	false
contextMenuBehaviour	inherit*
contextMenuId	
enabled	true
explanation	
resource	fileResource
state	normal*
textDirection	inherit*
tooltip	Upload File!
visible	visible*
width	

Now you have to implement the Action Event Handler *onActionUploadFile()*. After the *UploadFile* action has been triggered on the user interface, the chosen file is transported to the context attribute *fileResource* of type *IWDResource*. All its metadata can be accessed by invoking the *IWDResource-API*.

```
public void onActionUploadFile(IWDCustomEvent wdEvent){
    //@@begin onActionUploadFile(ServerEvent)
    IPrivateFileUploadView.IContextElement element =
        wdContext.currentContextElement();
    if (element.getFileResource() != null) {
        // if a file in the FileUpload field exists fill the context with
        // the file details, make the details visible and report
        // the success.
        IWDResource resource = element.getFileResource();
        element.setFileSize(this.getFileSize(resource));
        element.setFileExtension(
            resource.getResourceType().getFileExtension());
        element.setFileName(resource.getResourceName());
    }
}
```



```

    element.setDetailsVisibility(WDVisibility.VISIBLE);
    wdComponentAPI.getMessageManager().reportSuccess(
        wdComponentAPI.getTextAccessor().getText(
            IMessageTutorial.SF_UPLOAD,
            new Object[] { resource.getResourceName()}));
} else {
    // if no file in the FileUpload field exists hide the details and
    // report an error.
    element.setDetailsVisibility(WDVisibility.NONE);
    wdComponentAPI.getMessageManager().reportException(
        wdComponentAPI.getTextAccessor().getText(
            IMessageTutorial.NO_FILE));
}
// Note:
// We clear the FileUpload context value attribute because the
// resource is not needed afterwards anymore in this application.
// Instead it is possible to upload another file.
element.setFileResource(null);
//@@end
}

```

The size of the retrieved MIME resource is calculated within the private method `getFileSize()` which is added to the last user coding area between the lines `//@begin` and `//@end`:

```


/**
 * Read resource and calculate file size.
 * @return the file size in Bytes, KB or MB as String
 */
private String getFileSize(IWDRResource resource) {
    InputStream stream = null;
    DecimalFormat myFormatter = new DecimalFormat("###.##");
    double size = 0;
    String unit = "";
    try {
        stream = resource.read(false);
        int length = 0;
        byte[] part = new byte[10240];
        while ((length = stream.read(part)) != -1) {
            size += length;
        }
        if (size < 1024) {
            unit = " Bytes";
        } else if (size < 1048576) {
            size = size / 1024;
            unit = " KB";
        } else if (size < 1073741824) {
            size = size / 1024 / 1024;
            unit = " MB";
        }
    } catch (IOException e) {
        wdComponentAPI.getMessageManager().reportException(e);
    } finally {
        if (stream != null) {
            try {
                stream.close();
            } catch (IOException e) {

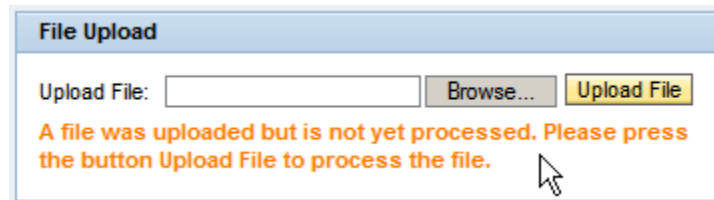
```

```

        wdComponentAPI.getMessageManager().reportException(e);
    }
}
}
return myFormatter.format(size) + unit;
}
}

```

 Note that the selected file in the *FileUpload* UI element (after using the Browse button) is uploaded **automatically with the next server roundtrip** independently from the user action which causes the roundtrip (e.g. pressing the right mouse button to open a context menu). The *FileUpload* UI element is cleared again after the roundtrip to the server. Our recommendation is to give the user an explicit signal that a file was uploaded successfully to avoid confusions. The tutorial application shows for example a message when a file was selected with the Browse button but then the users tries to open a context menu via the right mouse button:

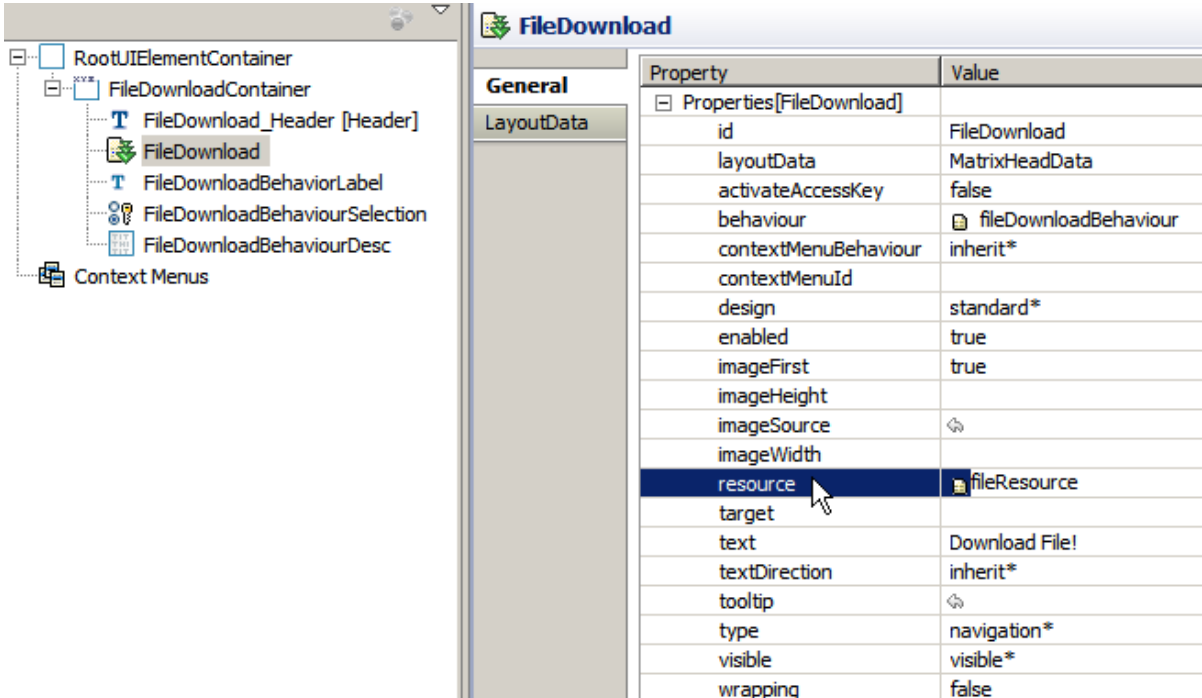


Downloading Files


This section describes the implementation details of the *FileDownloadView*.


When the file is downloaded, a MIME file saved in the context is downloaded by the client. To do this, first define a new context attribute of dictionary simple type **Resource** in the view context.

After defining the *Resource*-type context attribute *fileResource*, you can bind a new *FileDownload* UI element to it. This UI element is displayed as a link on the user interface. When this link is selected, the MIME file saved in the context attribute is automatically transported to the user interface. With the transfer of the associated MIME type, the connected software on the client side can be started in order to view the MIME file.



Property	Value
Properties[FileDownload]	
id	FileDownload
layoutData	MatrixHeadData
activateAccessKey	false
behaviour	fileDownloadBehaviour
contextMenuBehaviour	inherit*
contextMenuId	
design	standard*
enabled	true
imageFirst	true
imageHeight	
imageSource	
imageWidth	
resource	fileResource
target	
text	Download File!
textDirection	inherit*
tooltip	
type	navigation*
visible	visible*
wrapping	false

 Web Dynpro defines `_blank` as the default value for the property `target` in the UI elements `FileDownload` and `LinkToURL`. Although the `target` values `_top`, `_parent`, and `_self` can be used, you are advised not to do so, as this will cause the Web Dynpro application to crash. Whereas `target` value `_blank` always causes a new target window to open, entering a `target` value like `"SameWindow"` causes a downloaded file to always open in the same target window (not the window of the application) if the UI element `FileDownload` is clicked more than once.


 Note that no events are bound to an action when using UI element `FileDownload`. Nevertheless the downloadable resource can be retrieved from the server on-demand, this means after it is requested by the user. This on-demand streaming technique is elaborately described in the `TableView` section.

In addition you can dynamically set the `behavior` property of the `FileDownload` UI element. The enumeration type `WDFileDownloadBehaviour` determines how the downloaded file is represented on the client. This property can have three different values:

- **AUTO**: The behaviour is predefined and depends on the mime type of the downloaded file.
- **ALLOW_SAVE**: An open/save dialog asks the user.
- **OPEN_INPLACE**: The file will be opened in place in the web page with the browser-embedded application program.

The property `behavior` of the `FileDownload` UI element is bound to the context attribute `fileDownloadBehaviour` which has the corresponding `SimpleType FileDownloadBehaviour`.

The controller implementation required for the file download is in this case restricted to method `wdDoInit()`, which is called when the view controller is created. Unlike file upload, no action event handling takes place after the download process has been triggered by the user. The MIME file selected for downloading is therefore stored in the context at the controller initialization stage.

 When using the `FileDownload` UI element as a table cell editor you should apply the on-demand stream technique which is described in the `TableView` section. With this approach you must not initially store all resources in the context before the user actually requests them on client side.

To allow you to download a file in the example application, the project contains the image file `Sap.jpg`. This image file is stored in the project directory `src → mimes → Components → com.sap.test.tc.wd.tut.file.updwld.wd.comp.tutorial.Tutorial` and is deployed in the project archive on the SAP J2EE Server.

In the method `wdDoInit()` an `IWDRResource` is created for this file and attached to the context.

```
public void wdDoInit()
{
    //@@begin wdDoInit()

    IWDRResource resource = null;
    try {
        // The image file 'Sap.jpg' is deployed with the Web Dynpro project
        // (under src/mimes/Components...). The resource path (URL) for this
        // mime objects can be accessed using the WDUURLGenerator service.
        String resourcePath = WDUURLGenerator.getResourcePath(
            wdComponentAPI.getDeployableObjectPart(),
            FileDownloadView.FILE_NAME);
        // retrieve resource object for given resource path and create a new
        // object of type IWDRResource by invoking the WDRResourceFactory API.
        resource =
            WDRResourceFactory.createResource(
                new FileInputStream(new File(resourcePath)),
                FileDownloadView.FILE_NAME,
                FileDownloadView.FILE_EXT,

```

```

        true); //true: flush file input stream so that it gets closed
            //immediately
    } catch (WDAliasResolvingException e) {
        wdComponentAPI.getMessageManager().reportException(e);
    } catch (FileNotFoundException e) {
        wdComponentAPI.getMessageManager().reportException(
            wdComponentAPI.getTextAccessor().getText(
                IMessageTutorial.FNF, new Object[] {FILE_NAME}));
    }

    wdContext.currentContextElement().setFileResource(resource);

    // initialize the default file download behavior
    wdContext.currentContextElement().
        setFileDownloadBehaviour(WDFileDownloadBehaviour.AUTO);
    //@@end
}

```

The name of the image file `Sap.jpg` and its MIME type are stored in the two controller class constants `FILE_NAME` and `FILE_EXT`:

```

// store image file name and file extension in member constants
private static final String FILE_NAME = "Sap.jpg";
private static final WDWebResourceType FILE_EXT =
    WDWebResourceType.JPG_IMAGE;

```

Extended Upload and Download features

This section describes the implementation details of the *TableView*. In this section 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 two views we now find answers on the following technical questions:

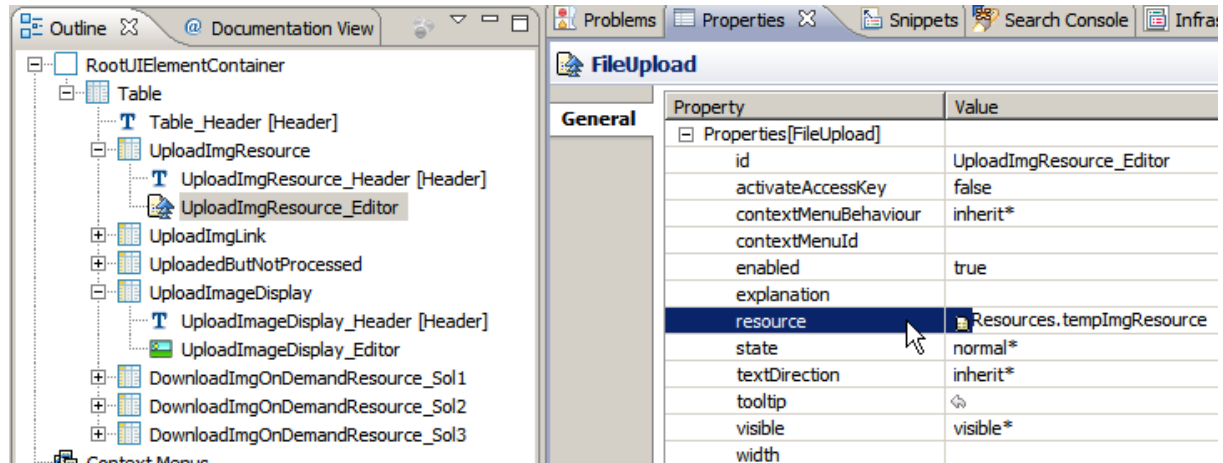
- How to upload image resources with different mime types (png, gif, jpg etc.) per table line?
- How to store these image resources in a multiple context node of the controller context?
- How to instantly display an uploaded image in a table cell?
- How to stream resource content (binary resource data)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

Uploading Files in a Table

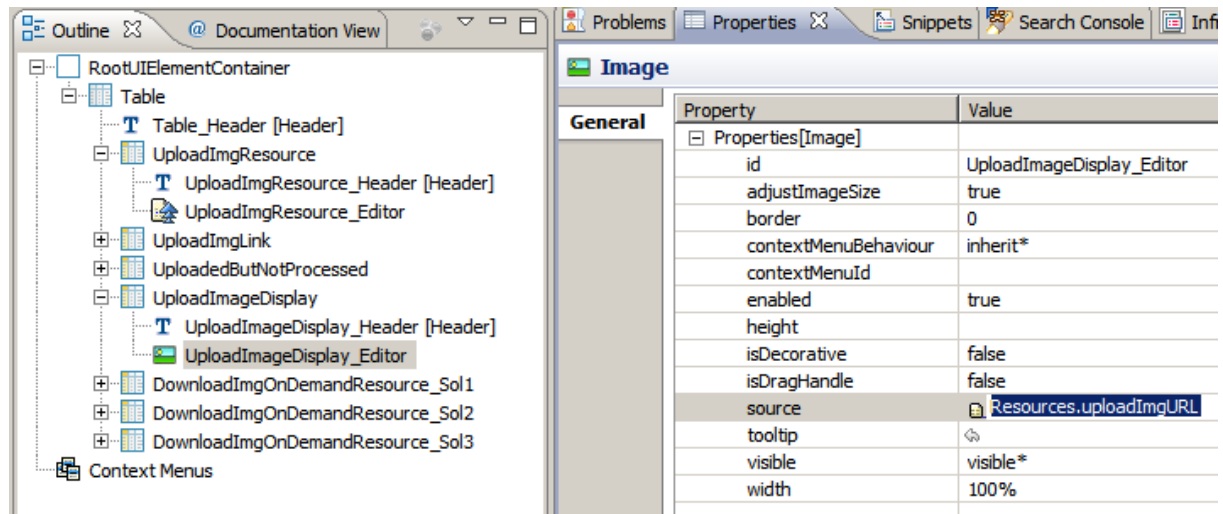
Within this section we demonstrate how to realize a simple use case of a *FileUpload* table cell editor:

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.

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 *templmgResource* of dictionary type *com.sap.ide.webdynpro.uelement.definitions.Resource*. 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 calculated context attribute *uploadImgURL* of type *String*. 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`.



At runtime all *Image* UI elements in table column *Image* are bound to the calculated context attribute *uploadImgURL* of the corresponding context node element. 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**. 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.

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

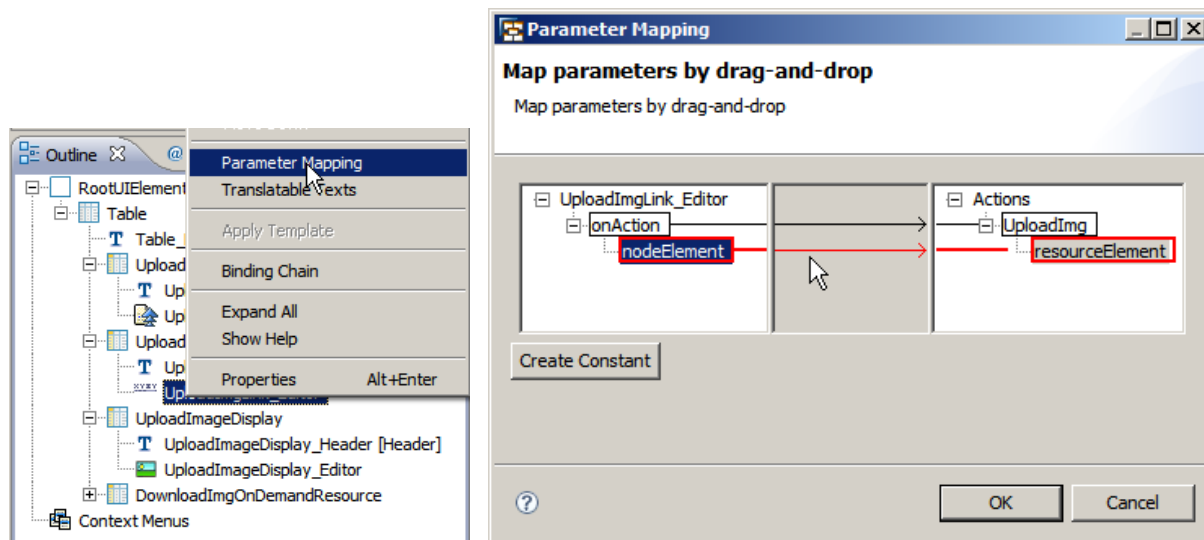
- **Name:** *UploadImg*
- **Text:** *Upload*. This text will be displayed in every line of table column *Upload*.

- **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.

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

```
public void wdDoModifyView(IWView view, boolean firstTime){
    //@@begin wdDoModifyView
    if (firstTime) {
        IWLinkToAction linkToAction =
            (IWLinkToAction) view.getElement("UploadImgLink_Editor");
        linkToAction.mappingOfOnAction().
            addSourceMapping("nodeElement", "resourceElement");
        //...
    }
    //@@end
}
```

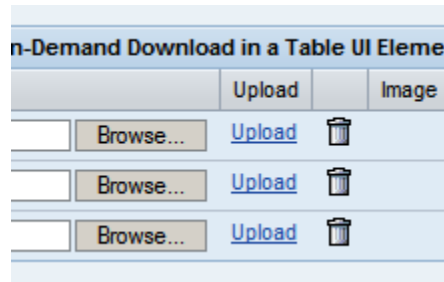
Instead of doing the parameter mapping programmatically it is also possible to do it in a declarative way. For this open the context menu on the `LinkToAction` UI element in the `Outline` view and select the entry `Parameter Mapping`.



Within the action event handler `onActionUploadImg()` we copy the temporary uploaded resource (context attribute `tempImgResource`) to the context attribute `uploadImgResource` of the same type and clear the context attribute `tempImgResource`.

```
public void onActionUploadImg(
    IWCustomEvent wdEvent,
    IPrivateTableView.IResourcesElement resourceElement){
    //@@begin onActionUploadImg(ServerEvent)
    if (resourceElement.getTempImgResource() != null) {
        // If a file was uploaded, move it from the temporary context
        // attribute to the final one.
        resourceElement.setUploadImgResource(
            resourceElement.getTempImgResource());
        resourceElement.setTempImgResource(null);
    }
    //@@end
}
```

This step is needed for the case when a file was uploaded within a roundtrip which is not caused by the *Upload* Link (see chapter *Uploading Files* above). The file is uploaded and accessible via the context attribute *tempImgResource* and the *File Upload* UI element is cleared after the roundtrip. To inform the user that the file was uploaded but not yet processed by the application (the action event handler `onActionUploadImg()` was not yet called) the tutorial application shows a trash icon in the corresponding table row. The icon is always visible when the *tempImgResource* context attribute is set. By clicking on the icon the temporary resource is cleared again.



The calculation of the URL of the uploaded image resource is done in the getter method `getResourcesUploadImgURL()` of the calculated context attribute *uploadImgURL*.

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`.

```
public java.lang.String getResourcesUploadImgURL(
    IPrivateTableView.IResourcesElement element){
    //@@begin getResourcesUploadImgURL
    if (element.getUploadImgResource() != null) {
        // if a uploaded file exists return the URL string from IWDResource.
        // The Image gets visible on the UI based on databinding definition.
        return element.getUploadImgResource().
            getUrl(WDFileDownloadBehaviour.OPEN_INPLACE.ordinal());
    }

    return null;
    //@@end
}
```

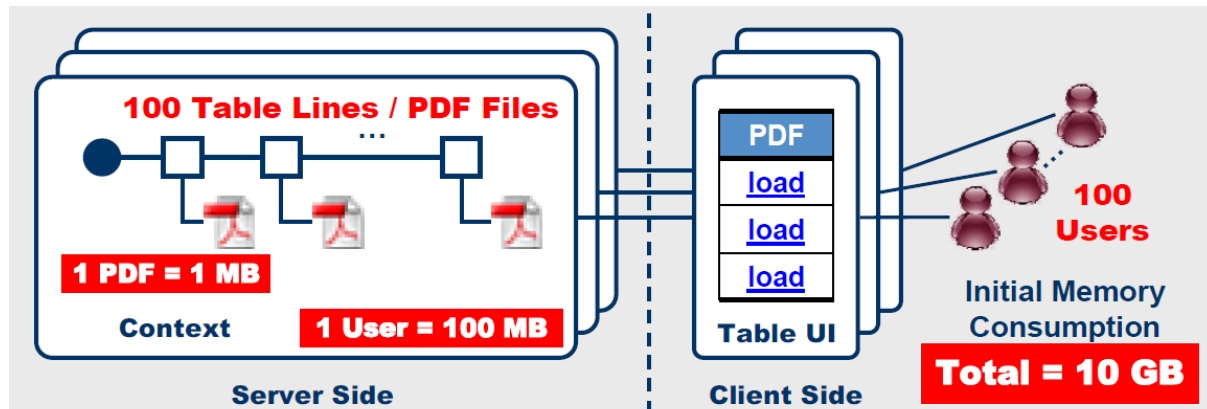
Downloading Files in a Table Using On-Demand Resources

Why using On-Demand Resources?

When using a *FileDownload* UI element as table cell editor in **SAP NetWeaver 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.



Please note, that the technical description of this example is based on **SAP NetWeaver 04**. Afterwards we will present solutions which can only be implemented in **SAP NetWeaver 04s** and later (**SAP NetWeaver 7.1***) using *on-demand Resource creation*.



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 `FileDownload` 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 `FileDownload` 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 1: Using On-Demand Streams and Calculated Context Attribute](#)

Since **SAP NetWeaver 04s** a function is available in Web Dynpro which solves the above memory consumption problem. But this solution has the disadvantage that a file handle is created and hold for a longer time. For **Net Weaver 7.1** and later the On-Demand Solutions 2 or 3, which are described below, should be used.

- **Context Attribute of type Resource:** The `FileDownload` UI element must be bound to a context attribute of type `com.sap.ide.webdynpro.uelementdefinitions.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*).
- **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.

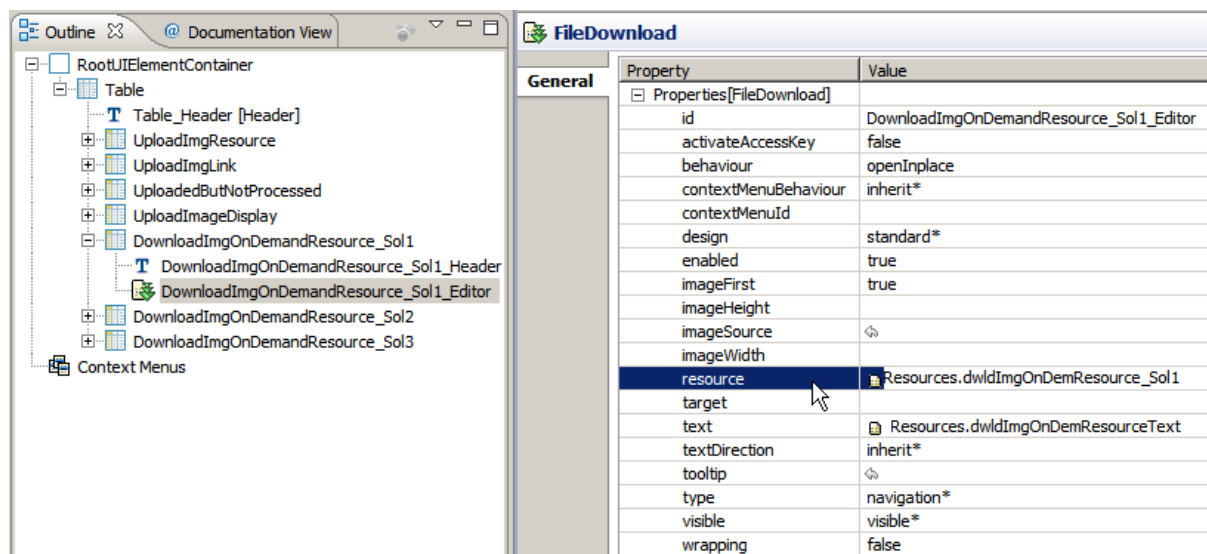
The process of *0-byte resource creation* doesn't initially allocate memory for the file content of the resource object but only for the file metadata. It enables the streaming of this content to be put off until it is actually requested by the client. Instead of being streamed to the context when the 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).

This solution is realized for the table column "On-Demand Download (1)".

The `FileDownload` UI element (`DownloadImgOnDemandResource_Sol1_Editor`) property `resource` is bound to the context attribute `dwldImgOnDemResource_Sol1` of dictionary type `Resource`.

Additionally a read-only *calculated* context attribute `dwldImgOnDemStreamCalc` of type `IWDInputStream` must be defined. There is no data binding relation between the `FileDownload` UI element and this calculated context attribute. The relation between the context attribute `dwldImgOnDemResource_Sol1` of type `IWDResource` to the calculated context attribute `dwldImgOnDemStreamCalc` of type `IWDInputStream` is implemented in the controller code.

When the user triggers a file download on client side 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.



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()`).

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 `WDRResourceFactory` class:

```
WDRResourceFactory.createResource(
    IWDAttributePointer attributePointer,
    java.lang.String resourceName,
    WDRWebResourceType 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()`.

```

public void supplyResources(
    IPrivateTableView.IResourcesNode node,
    IPrivateTableView.IContextElement parentElement) {
    //@@begin supplyResources(IWDNode,IWDNodeElement)
    IPrivateTableView.IResourceElement resourceElement;
    // ----- 1. Resource Node Element -----
    resourceElement =
        wdContext.nodeResources().createAndAddResourceElement();
    resourceElement.setDwldImgOnDemResourceText("SAP Logo - JPG");
    resourceElement.setDwldImgOnDemResourceName("SAPLogo.jpg");
    resourceElement.setDwldImgOnDemResourceType(
        WWebResourceType.JPG_IMAGE);
    resourceElement.setDwldImgOnDemResource_So11(
        this.create0ByteResource(resourceElement));
    //...
    // ----- 2. Resource Node Element -----
    resourceElement =
        wdContext.nodeResources().createAndAddResourceElement();
    resourceElement.setDwldImgOnDemResourceText("SAP NetWeaver - GIF");
    resourceElement.setDwldImgOnDemResourceName("SAPNetWeaver.gif");
    resourceElement.setDwldImgOnDemResourceType(
        WWebResourceType.GIF_IMAGE);
    resourceElement.setDwldImgOnDemResource_So11(
        this.create0ByteResource(resourceElement));
    //...
    // ----- 3. Resource Node Element -----
    resourceElement =
        wdContext.nodeResources().createAndAddResourceElement();
    resourceElement.setDwldImgOnDemResourceText("SAP TechEd - PNG");
    resourceElement.setDwldImgOnDemResourceName("SAPTechEd.png");
    resourceElement.setDwldImgOnDemResourceType(WWebResourceType.PNG);
    resourceElement.setDwldImgOnDemResource_So11(
        this.create0ByteResource(resourceElement));
    //...
    //@@end
}

/**
 * Returns a 0-byte resource object of type IWDResource which points to a
 * calculated 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(
    IResourceElement resourceElement) {
    return WResourceFactory.createResource(
        resourceElement.getAttributePointer(
            IResourceElement.DWLD_IMG_ON_DEM_STREAM_CALC),
        resourceElement.getDwldImgOnDemResourceName(),
        resourceElement.getDwldImgOnDemResourceType());
}

```

}

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*.

```

/**
 * Declared getter method for attribute 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
 */
/**
public com.sap.tc.webdynpro.progmodel.api.IWDInputStream
  getResourcesDwldImgOnDemStreamCalc(
    IPrivateTableView.IResourcesElement element){
  /**
  try {
    IWDResource resource =
      WDWebResource.getWebResource(
        wdComponentAPI.getDeployableObjectPart(),
        element.getDwldImgOnDemResource().getResourceType(),
        element.getDwldImgOnDemResource().getResourceName());
    InputStream inputStream = resource.read(false);
    return WDResourceFactory.createInputStream(inputStream);
  } catch (IOException e) {
    wdComponentAPI.getMessageManager().reportException(
      wdComponentAPI.getTextAccessor().getText(
        IMessageTutorial.FNF, new Object[] {
          element.getDwldImgOnDemResource().getResourceName()}));
    return null;
  }
  /**
}

```

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.



In this example all image resources are deployed with the tutorial component, stored under the folder `src → mimes → Components → com.sap.test.tc.wd.tut.file.updwld.wd.comp.tutorial.Tutorial`, so that they can be easily retrieved using the Web Dynpro services *WDURLGenerator* or *WDResourceFactory*. In a real Web Dynpro application scenario the file content must be retrieved

by executing a corresponding model object (Adaptive RFC, Adaptive Web Service) in the calculated context attribute getter method.

Solution 2: Using the *IWDResourceContentProvider* API

This solution can only be used in **SAP NetWeaver 7.1*** or later and is realized for the table column “On-Demand Download (2)”.

Similar to solution 1 the *FileDownload* UI element (*DownloadImgOnDemandResource_Sol2_Editor*) property *resource* is bound to the context attribute *dwldImgOnDemResource_Sol2* of dictionary type *Resource*.

Property	Value
id	DownloadImgOnDemandResource_Sol2_Editor
activateAccessKey	false
behaviour	openInplace
contextMenuBehaviour	inherit*
contextMenuId	
design	standard*
enabled	true
imageFirst	true
imageHeight	
imageSource	
imageWidth	
resource	Resources.dwldImgOnDemResource_Sol2
target	
text	Resources.dwldImgOnDemResourceText
textDirection	inherit*
tooltip	
type	navigation*
visible	visible*
wrapping	false

But for this solution no calculated attribute in the context is needed. Instead the resources are created in the context supply function `supplyResources()` using an instance of *IWDResourceContentProvider*.

To avoid code redundancy we encapsulate the creation of a resource which uses a Content Provider in the private controller method `createResourceUsingContentProvider()`.

```
public void supplyResources(
    IPrivateTableView.IResourcesNode node,
    IPrivateTableView.IContextElement parentElement) {
    //@begin supplyResources(IWDNode,IWDNodeElement)
    IPrivateTableView.IResourceElement resourceElement;
    // ----- 1. Resource Node Element -----
    //...
    resourceElement.setDwldImgOnDemResource_Sol2(
        this.createResourceUsingContentProvider(resourceElement));
    //...
    // ----- 2. Resource Node Element -----
    //...
    resourceElement.setDwldImgOnDemResource_Sol2(
        this.createResourceUsingContentProvider(resourceElement));
    //...
    // ----- 3. Resource Node Element -----
    //...
    resourceElement.setDwldImgOnDemResource_Sol2(
        this.createResourceUsingContentProvider(resourceElement));
    //@end
}
```

```

/**
 * Returns an object of type IWDResource based on an
 * <code>IWDResourceContentProvider</code> which provides the
 * file content on-demand.
 */
private IWDResource createResourceUsingContentProvider (
    IResourcesElement resourceElement) {
    return WDRResourceFactory.createResource (
        new DemoResourceContentProvider (
            resourceElement.getDwldImgOnDemResourceName ()),
        resourceElement.getDwldImgOnDemResourceName (),
        resourceElement.getDwldImgOnDemResourceType ());
}

```

The Web Dynpro API interface *IWDResourceContentProvider* must be implemented by the application (see class *DemoResourceContentProvider* in the custom coding area at the end of the *TableView* controller). The most interesting method of this interface is *writeContent* (*OutputStream outputStream*) which is called when the binary data is requested for download. The test application reads here the requested image resource which is deployed with the tutorial component and writes the read bytes into the provided output stream.

```

/**
 * <code>IWDResourceContentProvider</code> implementation which provides
 * the binary data for an <code>IWDResource</code> when needed. A
 * resource based on this interface does not copy the data to the binary
 * cache unless they are requested from the client.
 */
private class DemoResourceContentProvider
    implements IWDResourceContentProvider{
    private String resourceName;
    private long totalLength = -1;

    public DemoResourceContentProvider(String resourceName){
        this.resourceName = resourceName;
    }

    /**
     * @see com.sap.tc.webdynpro.services.sal.datatransport.api.
     * IWDResourceContentProvider#getLength()
     */
    public long getLength() {
        return totalLength;
    }

    /**
     * @see com.sap.tc.webdynpro.services.sal.datatransport.api.
     * IWDResourceContentProvider#isAvailable()
     */
    public boolean isAvailable() {
        return true;
    }

    /**
     * @see com.sap.tc.webdynpro.services.sal.datatransport.api.
     * IWDResourceContentProvider#writeContent (java.io.OutputStream)
     */

```

```

public void writeContent(OutputStream outputStream) throws IOException
{
    InputStream inputStream = null;
    try {
        String resourcePath = WDURLGenerator.getResourcePath(
            wdComponentAPI.getDeployableObjectPart(), resourceName);
        inputStream = new FileInputStream(new File(resourcePath));

        totalLength = write(inputStream, outputStream);

    } catch(WDAliasResolvingException e) {
        wdComponentAPI.getMessageManager().reportException(e);
    } finally {
        if (inputStream != null) {
            try {
                inputStream.close();
            } catch (IOException e) {
                wdComponentAPI.getMessageManager().reportException(e);
            }
        }
    }
}

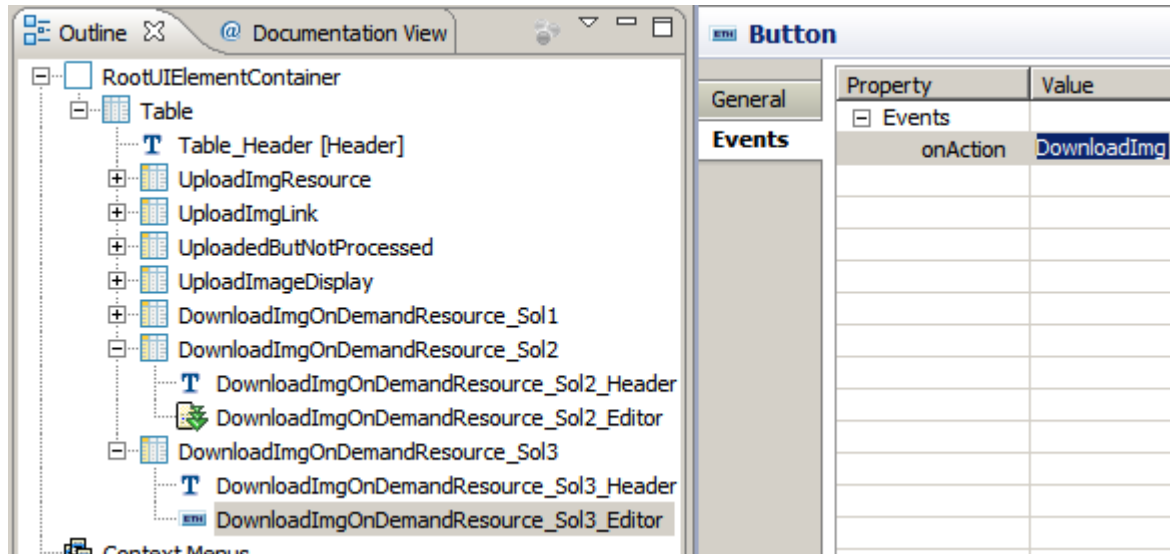
/**
 * Writes the input stream in the output stream.
 * Both streams are not closed.
 *
 * @return sum of written bytes
 */
private long write(InputStream in, OutputStream out)
    throws IOException{
    int length = 0;
    long totalLength = 0;
    byte[] part = new byte[10240];
    while ((length = in.read(part)) != -1) {
        out.write(part, 0, length);
        totalLength += length;
    }
    return totalLength;
}
}

```

Solution 3: Using the `IWDResource.download()` API

This solution can only be used in **SAP NetWeaver 7.1*** or later and is realized for the table column “On-Demand Download (3)”.

In contrast to the other solutions no `FileDownload` UI element is used here. Instead the cell editor for the column “On-Demand Download (3)” is a `Button`. The `onAction` event of this button is bound to the Action `DownloadImg`. The corresponding action event handler method is `onActionDownloadImg()`. All the necessary code for the download is placed there.



For the button also the parameter mapping technique is used to automatically pass the action-triggering node element to the action event handler. This technique is already described above in detail and can also be done declaratively.

```
public void wdDoModifyView(IWView view, boolean firstTime){
    //@@begin wdDoModifyView
    if (firstTime) {
        //...
        IWDButton button = (IWDButton) view.getElement(
            "DownloadImgOnDemandResource_Sol3_Editor");
        button.mappingOfOnAction()
            .addSourceMapping("nodeElement", "resourceElement");
    }
    //@@end
}
```

In the action event handler method `onActionDownloadImg()` a new resource is created for the requested image which is deployed with the tutorial component when the resource is not yet available in the context (when the download is triggered for the first time).

This new resource is cached in the context (attribute `dwldImgOnDemResource_Sol3`) and is reused when the download is triggered a second time. Holding the resource is also important to prevent the Java VM Garbage Collector to destroy the resource instance after the execution of the method because then also the computed download URL will become invalid.

Finally the method `download()` is called on the resource to start the download.

```
public void onActionDownloadImg(IWDCustomEvent wdEvent,
    IPrivateTableView.IResourcesElement resourceElement) {
    //@@begin onActionDownloadImg(ServerEvent)
    IWDResource resource = resourceElement.getDwldImgOnDemResource_Sol3();
    if(resource == null){
        try {
            // The image file is deployed with the Web Dynpro project
            // (under src/mimes/Components...). The resource path (URL) for
            // this mime objects can be accessed using the WDURLGenerator
            // service.
            String resourcePath = WDURLGenerator.getResourcePath(
                wdComponentAPI.getDeployableObjectPart(),
                resourceElement.getDwldImgOnDemResourceName());
        }
    }
}
```

```
// retrieve resource object for given resource path and create
// a new object of type IWDResource by invoking the
// WDRResourceFactory API.
resource =
    WDRResourceFactory.createResource(
        new FileInputStream(new File(resourcePath)),
        resourceElement.getDwldImgOnDemResourceName(),
        resourceElement.getDwldImgOnDemResourceType(),
        true); //true: flush file input stream so that it gets closed
              // immediately
} catch (WDAliasResolvingException e) {
    wdComponentAPI.getMessageManager().reportException(e);
} catch (FileNotFoundException e) {
    wdComponentAPI.getMessageManager().reportException(
        wdComponentAPI.getTextAccessor().getText(
            IMessageTutorial.FNF,
            new Object[]{resourceElement.getDwldImgOnDemResourceName()}
        ));
}

// Store the resource for later reuse and to ensure
// that it is not immediately cleaned up by the JVM Garbage
// Collector.
resourceElement.setDwldImgOnDemResource_So13(resource);
}

if(resource != null){
    resource.download();
}
}
```


Further Information



SAP NetWeaver CE 7.1 EHP1 Help (File Upload and File Download)

http://help.sap.com/saphelp_nwce711/helpdata/en/42/dfd9c528d45171e10000000a1553f7/frameset.htm



SAP NetWeaver CE 7.1 EHP1 Help (Delivered Virus Scan Profiles)

http://help.sap.com/saphelp_nwce711/helpdata/en/21/479a4271c80a31e10000000a1550b0/frameset.htm







This tutorial is based on the following File Upload and File Download tutorial and article which are available in SDN:

Tutorial: <http://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/00062266-3aa9-2910-d485-f1088c3a4d71>

Article: <http://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/b0e10426-77ca-2910-7eb5-d7d8982cb83f>

Legend

Text Symbols

Symbol	Usage
	Note
	Recommendation
	Warning
	See also

Copyright

© Copyright 2010 SAP AG. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft, Windows, Excel, Outlook, and PowerPoint are registered trademarks of Microsoft Corporation.

IBM, DB2, DB2 Universal Database, System i, System i5, System p, System p5, System x, System z, System z10, System z9, z10, z9, iSeries, pSeries, xSeries, zSeries, eServer, z/VM, z/OS, i5/OS, S/390, OS/390, OS/400, AS/400, S/390 Parallel Enterprise Server, PowerVM, Power Architecture, POWER6+, POWER6, POWER5+, POWER5, POWER, OpenPower, PowerPC, BatchPipes, BladeCenter, System Storage, GPFS, HACMP, RETAIN, DB2 Connect, RACF, Redbooks, OS/2, Parallel Sysplex, MVS/ESA, AIX, Intelligent Miner, WebSphere, Netfinity, Tivoli and Informix are trademarks or registered trademarks of IBM Corporation.

Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.

Adobe, the Adobe logo, Acrobat, PostScript, and Reader are either trademarks or registered trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Oracle is a registered trademark of Oracle Corporation.

UNIX, X/Open, OSF/1, and Motif are registered trademarks of the Open Group.

Citrix, ICA, Program Neighborhood, MetaFrame, WinFrame, VideoFrame, and MultiWin are trademarks or registered trademarks of Citrix Systems, Inc.

HTML, XML, XHTML and W3C are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.

Java is a registered trademark of Sun Microsystems, Inc.

JavaScript is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

SAP, R/3, SAP NetWeaver, Duet, PartnerEdge, ByDesign, SAP Business ByDesign, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and other countries.

Business Objects and the Business Objects logo, BusinessObjects, Crystal Reports, Crystal Decisions, Web Intelligence, Xcelsius, and other Business Objects products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of Business Objects S.A. in the United States and in other countries. Business Objects is an SAP company.

All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serves informational purposes only. National product specifications may vary.

These materials are subject to change without notice. These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.