



Building ColdFusion Builder Extensions

Ram Kulkarni,
Sr. Computer Scientist,
Adobe



Agenda

- What is ColdFusion Builder Extension
- Demo: CFC Generator
- Developing Extensions
 - Extension Metadata
 - Installer Wizard
 - Project Contribution
 - Menu Contributions
 - Generating UI from handler
 - Extension in View
 - Contributing to Code Assist
 - Callback Commands
- Using Extension Builder

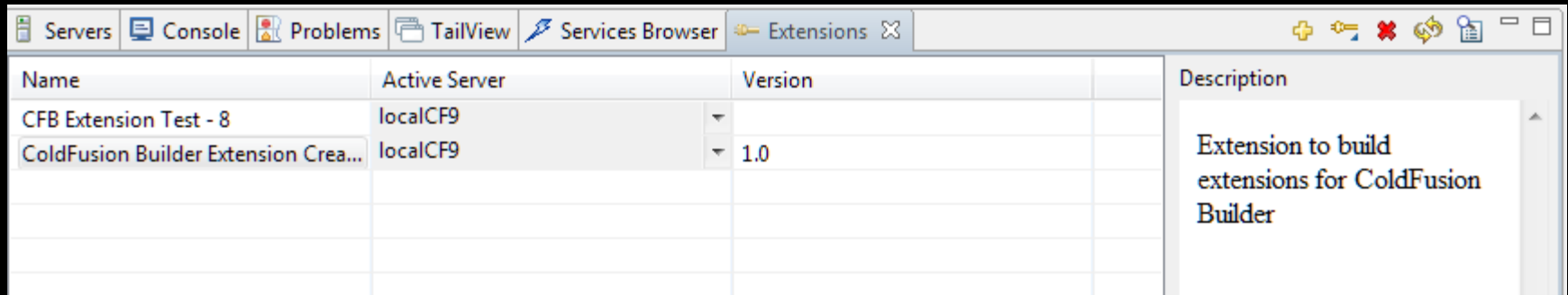
ColdFusion Builder Extensions






- Extend functionality of ColdFusion Builder
- Write extensions using CFML
- Extension is executed on the Server

How CFB Extension works

- Extension is installed on the server and configured in the Builder
- Lets you add menu options in
 - Navigator
 - RDS Data View
 - Outline View
 - Project Wizard
- Write Handler CFM for each menu option
- Invoking menu option executes Handler CFM on the server
- Handler CFM can
 - Create HTML UI
 - Send commands back to ColdFusion Builder to perform some actions

Extension View in ColdFusion Builder



- Extension view lists all installed extensions
- Install extension by clicking  icon
- Import extension that is already in wwwroot by clicking 
- To delete an extension. Select it in the extensions view and click 
- If you changed ide_config.xml (e.g. during development), you need to refresh extensions by clicking 
- You can see details of selected extension by clicking 

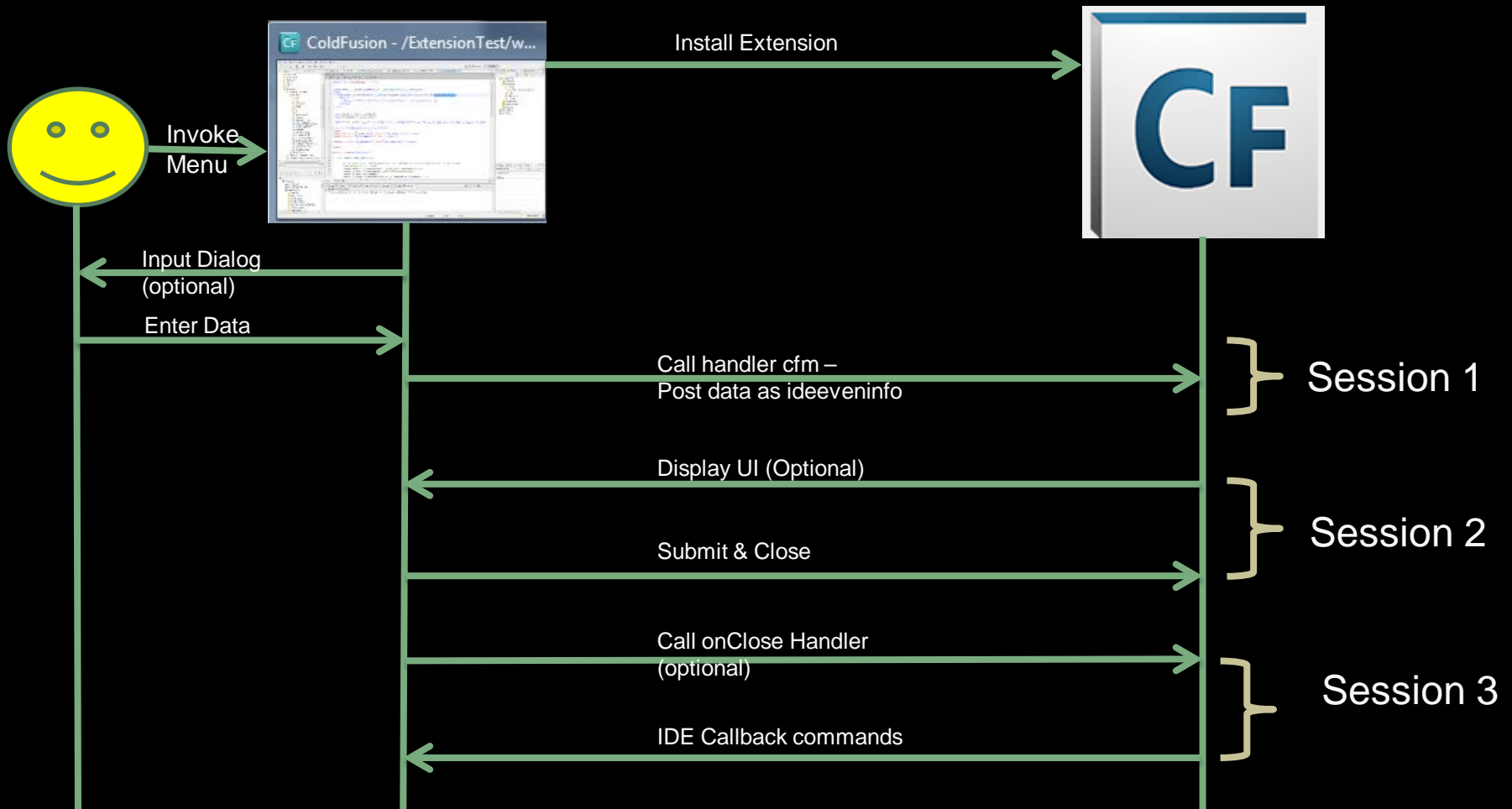


Demo

ORM CFC Generator



ColdFusion Builder Extension



Extension Zip File:

- **Extension.zip**

- handlers: all handler cfm files go in this folder
- ide_config.xml: Extension configuration file

- **ide_config.xml**

- `<application>`

```
    <!--Extension Info -->
```

```
    <name>Name of the Extension</name> <!-- required -->
```

```
    <author>Name of author(s)</author>
```

```
    <email></email>
```

```
    <description></description>
```

```
    <!-- License, optional -->
```

```
    <license>License text or name of the file in  
handlers folder </license>
```

```
</application>
```



Demo – Extension1

Create an Extension with
only extension info in
ide_config.xml



ide_config.xml – Installer Input

Adding input pages to extension installer :

```
<application>
  <!-- Application -->
  <!-- License -->

  <!-- Installer -->
  <wizard height="300" width="500" handlerid="installHandlerId">
    <page title="First Input Page">
      <input name="" label="" type="" required="true">
        <!-- more fields -->
      </input>
    </page>
    <!-- more pages -->
  </wizard>
  <!-- Handlers -->
  <handlers>
    <handler id="installHandlerId" type="cfm"
filename="install.cfm">
  </handlers>

</application>
```

Input Field Attributes :

| Attribute | Description |
|--------------|--|
| name | Name of the input field |
| label | Input field label |
| tooltip | Tip on mouse hover |
| type | dir, string, boolean, file, password, projectdir, projectfile, and list* |
| required | true / false |
| pattern | RegEx for validating input data |
| errormessage | Displayed when validation fails |
| helpmessage | Displayed in the title area |
| default | Default value for the given input type |
| checked | Applicable if type is boolean |

* If type is 'list' then input field can have `option` tags. For example,

```
<input type="list" name="list1">  
  <option name="opt1" value="opt1">  
</input>
```

ideeventinfo - Input XML to Extension Handler

```
<event>
  <ide version="2.0">
    <callbackurl>

      http://localhost:59355/index.cfm?extension=extension_name
    </callbackurl>
  </ide>

  <!-- event specific information goes here -->

  <user> <!-- user defined fields -->
    <input name="inputname" value="value" />
  </user>
</event>
```



Demo – Extension2

Create an Extension with
Installer Input



ide_config.xml: Events

```
<application>
  <!-- Application -->
  <!-- License -->
  <!-- Installer -->

  <!-- Events -->
  <events>
    <event type="onprojectcreate"
handlerid="projectHandlerId">

  </events>

  <!-- Handlers -->
  <handlers>
    <handler id="projectHandlerId" type="cfm" filename="onProject.cfm">
  </handlers>
</application>
```



Demo – Extension3

Create a project from a template using the **onProjectCreate** event



ide_config.xml: Menu Contributions

```
<application>
  <!-- Application, License, Installer -->

  <MenuContributions>
    <contribution target="rdsview, projectview, outlineview or
editor" >
      <menu name="top_level_menu_name">
        <!-- Menu Items -->
        <action name="menu1" handlerId="menuHandlerId"
showResponse="true/false">
          <input .../>
          <input .../>
        <!-- Menu Filters -->
        <filters>
          <filter type="file" pattern="orm*.cfm">
            <!-- filter types can be file, folder or project -->
          </filter>
        </filters>
      </menu>
    </contribution>
  </MenuContributions>

  <!-- Handlers -->
</application>
```

Keywords for Default Values of Input Fields

- `projectlocation`
- `projectname`
- `serverhome`
- `wwwroot`

Example :

```
<input name="location" label="Project Path" type="dir"  
  default="{ $projectlocation} ">
```

Creating Dialog for Input Fields

```
<dialog width="500" height="400" title="Dialog Title"  
  image="path_from_extention_folder">  
  
  <input ...>  
  
</dialog>
```

- Dialogs are not necessary for inputs
- Use dialog when you specify
 - Title
 - Image
 - Width and height



Demo – Extension4

Menu contribution to
RDSView, Navigator, and
OutlineView



Creating UI from Extension

- Set `showresponse` attribute to `true` in `action` tag in `ide_config.xml`
- Set content type in Handler CFM to `text/xml`
- Create the following XML response from the Handler CFM:

```
<response showresponse="true">
  <ide url="full_path_of_redirection_cfm">
    <dialog width="" height="" />
  </ide>
</response>
```

- Create UI in the redirected CFM
- (Optional) Create `application.cfm/cfc` in the handler folder to pass `ideeventinfo` to the redirected CFM

Sample code for creating response -

```
<cfparam name="ideeventinfo" >
<cfheader name="Content-Type" value="text/xml">
<cfset application.ideeventinfo = ideeventinfo>
<cfoutput>
    <response showresponse="true" >
        <ide url="complete_url_of_cfm_page">
            <dialog width="600" height="500" title="Sample
Response" />
        </ide>
    </response>
</cfoutput>
```



Demo – Extension5

Create UI from the Handler
CFM file



Running Extension in a View

- By default, extensions run in a modal dialog box
- Sometimes you need to refer to both extension and other views/editors in ColdFusion Builder
- ColdFusion Builder 2 lets you run extension in a view
- Two ways to run extension in a view:
 - In response to invoking extension menu actions
 - Invoking from the Eclipse option Window >Show View menu

Configuring View in IDE_config.xml

```
<application>
  <!-- Application -->
  <!-- License -->
  <!-- installer -->

  <viewcontributions>
    <view id="viewid" title="title" icon=""
handlerid="">
      <toolbaritem icon="" handlerid="">
    </view>
  </viewcontributions>

  <!-- Handlers -->
</application>
```

Creating view from Handler CFM

```
<cfheader name="Content-Type" value="text/xml">
<cfset application.ideeventinfo = ideeventinfo>
<cfoutput>

    <response showresponse="true" >
        <ide url="">
            <view />
        </ide>
    </response>
</cfoutput>
```

Note : The `view` tag above can have attributes such as `title`, `id`, or `icon`. It can also have `toolbaritem` child tags



Demo – Extension6

Extension running in a View



Code Assist Contributions

- You can add proposals to Code Assist window using extensions
- Code Assist contribution could be for
 - Arguments of function; could be UDF or function call on a CFC
 - Members of any variable, for example functions or member variables
- Steps to contribute to Code Assist
 - Add Code Assist contribution tag in the IDE_config.xml
 - Specify function names and argument number for which extension adds Code Assist proposals
 - Specify variable for which extension adds Code Assist proposals
 - Specify handlers for Code Assist contributions
 - In the Handler CFM file, return valid code assist proposals in an XML response

Code Assist Contribution: IDE_config.xml

Configuring View in IDE_config.xml

```
<application>
  <!-- Application -->
  <!-- License -->
  <!-- installer -->

  <codeassistcontribution>
    <functions>
      <function name="funcName" variableName="" component=""
        handlerid="">
        <parameter index="1">
        </function>
      </functions>

      <variables>
        <variable name="varName" component="" handlerid="" />
      </variables>
    </codeassistcontribution>

    <!-- Handlers -->
  </application>
```

Returning CA proposals from handler

```
<cfheader name="Content-Type" value="text/xml">
<cfoutput>
  <codeassist_response>

    <proposal display="proposal1"
insert=' "proposal1" '>

    <proposal display="proposal2"
insert=' "proposal2" '>

  </codeassist_response>
</cfoutput>
```



Demo – Extension7

Code Assist Contribution



Callback Commands

- Callback command lets extension handler to run commands in ColdFusion Builder
- List of callback commands:

| | |
|---------------------------------------|----------------------------|
| <code>refreshFile</code> | <code>refreshFolder</code> |
| <code>refreshProject</code> | <code>openFile</code> |
| <code>insertText</code> | <code>getServers</code> |
| <code>getDatasources</code> | <code>getTables</code> |
| <code>getTable</code> | <code>searchFile</code> |
| <code>getFunctionsAndVariables</code> | |

- ColdFusion Builder 1 lets call back commands only on closing the extension window (using onclose handler)

ColdFusion Builder 2 lets call back commands to be invoked any time from the Handler CFM using callback URL

Creating commads xml in Handler CFM

```
<cfsavecontent variable="command" >
  <cfoutput>
    <response>
      <ide>
        <commands>
          <command type="">
            <param key="" value="">
              <!-- more params here -->
            </command>
          <!-- more commands -->
        </commands>
      </ide>
    </response>
  </cfoutput>
</cfsavecontent>
```

Sending commands using cfhttp

```
<cftry>
    <cfhttp url="#callbackURL#" method="post"
result="httpResult" >
        <cfhttpparam type="body" value="#commands#" >
    </cfhttp>
    <cfcatch >
        <!-- handle error here -->
    </cfcatch>
</cftry>

<!-- command response is in httpResult variable -->
```


Troubleshooting Extensions

- **Unable to install extension**
 - Incorrect packaging
 - Directory name is not unique
- **Clicking extension menu does not trigger any action**
 - Verify the URL, path, and server specifications
 - Handler does not match the corresponding action's handlerid
- **Using extension results in exceptions**
 - Verify the log in Error Log View of ColdFusion Builder
- **Page not found error when running extension in a view**
 - Make sure you specify complete URLs



Thank you

rakulkar@adobe.com

