

IBM Endpoint Manager

*Application Development
Framework – Version 2.3.0*

Developer's Guide





Note: Before using this information and the product it supports, read the information in Notices.

© Copyright IBM Corporation 2003, 2013.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.



Contents

Part One	1
Developer's Overview	1
Requirements	1
Application Types	2
Dashboards and Wizards	2
Web Reports	2
Development Process	2
Loading a Project into the Console	2
Debugging	3
Disabling the trustunsignedwizards warning	4
Using the ADF in a Project	4
Localization	4
Left/Domain Navigation	4
Dashboard Datastore	4
Key Considerations	5
Working with Nested Folders	5
How IE Runs Differently in the Console	5
Single Xlat File	6
Xlat tags Only Work in Top-level files	6
Sharing Data Across Dashboards/Wizards	6
Relevance Results	7
Application Display/Layout	7
File Size/Number Propagated	7
Using requireJS and build tools	7
Run the build script	7
Difference between dev and built versions	8
RequireJS config	8
Part Two	9
API	9
JavaScript API Files	9
JavaScript API Overview	9
tem	10
tem.config	10
tem.consoleUtils	10
tem.content	10
tem.context	10
tem.dataStore	10
tem.file	11
tem.javaScriptUtils	11
tem.l10n	11
tem.library	11
tem.model	11
tem.relevance	11
tem.relevance.queries	11



tem.shell	11
Non-JavaScript Hooks	11
Relevance Tag	12
The link: Protocol and Opening Content Windows	12
Xlat tags for localization	12
Namespacing	14
Part Three	16
UI	16
ADF UI (jQuery-based) vs Dojo	16
Look and Feel	16
Using the ADF UI Components	16
How to Use the Sample Project	16
How to Create a New Dashboard/Wizard	16
Critical Issues	16
UI Components/temui	17
Use of colors	18
Internal Localization	19
Default Application Initialization	19
Support for AngularJS	19
AngularJS support in console	19
ADF AngularJS components	20
Support for Dojo Dashboards	20
How to Use the Dojo Sample Project	20
Critical Issues	20
Dojo Patch	21
Library Install	21
Internal Localization	22
RelevanceResultsStore	22
Default Application Initialization	22
Part Four	23
Notices	23



Part One

Developer's Overview

The Application Development Framework (ADF) enables rapid, standardized development of rich applications in the form of dashboards and wizards that display within the TEM Console and web reports that display in a separate web-based interface. Version 2.0 of the ADF is a JavaScript-only release that presents a repackaged JavaScript API layer that can be used for development with any JavaScript based UI framework (or any UI framework that can work with JavaScript). In addition it provides its own set of UI components based on jQuery plugins and libraries. It also provides some code for supporting the development of Dojo applications for the Console.

The contents of this document are as follows:

- **Chapter 1: "Developer's Overview"** - a high-level overview of the applications developed with ADF, descriptions of the typical development process and a list of some of the main issues/concerns when developing any application for the Console or Web Reports.
- **Chapter 2: "API"** - describes the full set of backend functionality available for application development.
- **Chapter 3: "UI"** - describes set of UI components available, the sample applications provided and additional support code necessary for running applications as well as critical issues involved when building the UI.

Some information related to application development is outside of the scope of the ADF and not included in this document:

- **Session Relevance** - Documentation for TEM 8.0 Session Inspectors can be found at: http://support.bigfix.com/fixlet/documents/Session%20Inspectors%2091_110719.pdf
- **Domain Specifications** - Used to construct left navigation in the Console.

Requirements

	ADF Requirements	Notes
TEM Version	8.0+	At a basic level the ADF API and UI work with 8.0+ setups. Some functionality and features are available only in later versions as noted in the documentation.



Application Types

Dashboards and Wizards

ADF is used for developing dashboards or wizards that display in the TEM Console. Dashboards are typically a set of report "pages" that aggregate the results of analyses in the form of charts and datagrids. Wizards perform specific tasks, such as providing a way to create customized content (i.e. tasks, fixlets or actions) . Wizards can also provide a customized UI for something like running an external program or storing user data in the dashboard datastore (see ["Dashboard Datastore" in Part One](#)).

Dashboards and wizards get loaded by the Console in the same manner. Each dashboard or wizard is defined by an .ojo file, which is an XML file that defines metadata about a dashboard or wizard and also contains an HTML element. This HTML element gets rendered by the Console by a web browser control that borrows a number of settings from the user's IE browser settings (whether to debug, plugin versions etc.) though there are a number of differences as well, see ["How IE Runs Differently in the Console" in Part One](#). The .ojo's HTML is the UI that the user sees as the dashboard or wizard.

Web Reports

ADF is also used for developing web reports. Web reports are similar to dashboards in that they typically show the same type of report data in the form of charts and datagrids. Web reports differ in that instead of an .ojo file they are defined by a .webreport file. The .webreport file is similar to the .ojo file in that it is an XML file with metadata and an HTML element, but it is loaded by the web-based web reports application.

Web reports have access to most but not all of the same types of functionality and components as dashboards and wizards. There are cases where an individual component or utility function's behavior differs between the two. Where differences occur they are noted in the API documentation for the individual functions.

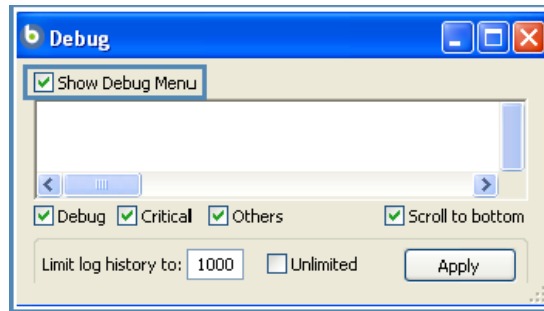
Development Process

During development, the dashboard/wizard is loaded into the Console via the Console's "Load Wizard..." dialog.

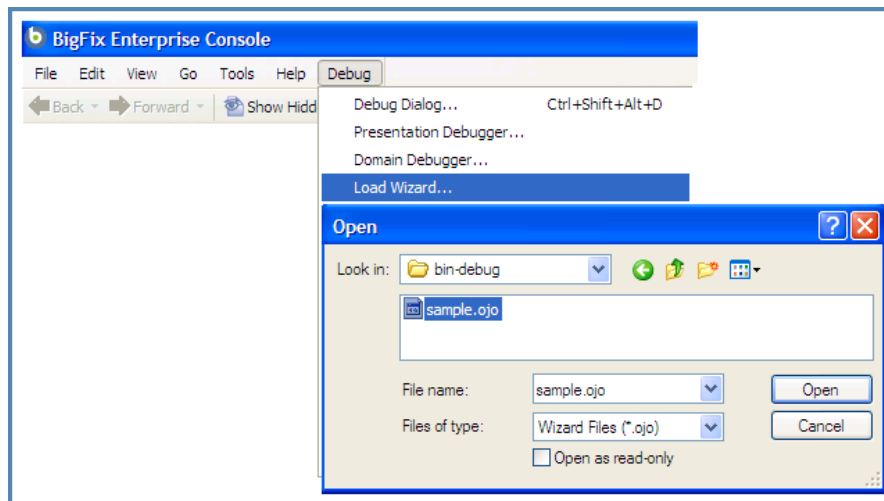
When the application is ready for a release, the application .ojo (or .webreport) along with all of its supporting files get added as part of a site, and ultimately get propagated with the site for use in a product.

Loading a Project into the Console

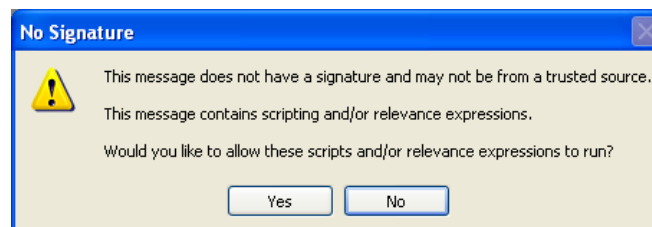
1. Setup the Console to display the debug menu:
 - a. This step needs to be done only once, when loading a project into the Console for the first time.
 - b. In the Console, Ctrl-Alt-Shift-D. In the Debug dialog, select Show Debug Menu, then Apply.



2. Load the compiled project into the Console.
 - a. Go to Debug > Load Wizard. From the directory that holds your project files (in this case /bin-debug), select your dashboard .ojo file, then Open.



- b. At this point, a signature warning appears; this warning can be disabled in the Registry (refer to [“Disabling the trustunsignedwizards warning”](#) section in [Part One](#)).



- c. To display the project in the Console, click Yes and the project starts to run in the Console.

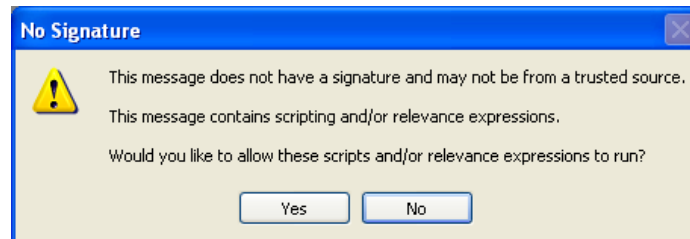
Debugging

As dashboards and wizards run in an IE instance in the Console, external JavaScript debuggers can be used such as Microsoft Script Editor or the debugger included with Visual Studio. These debuggers can be triggered by including the "debugger;" keyword in your JavaScript or when an error occurs (make sure IE script debugging is enabled.) Please note that the integrated debugger included with IE 8 cannot be used.

Web reports run directly in any browser so any JavaScript debugger utility can be used including the integrated debugger packaged with IE 8 and Firebug.

Disabling the trustunsignedwizards warning

While wizards loaded from a propagated site have a signed source, wizards loaded using the Load > Wizard mechanism (for example, for testing in the Console) are not signed and may display a warning:



Using the ADF in a Project

The ADF can be downloaded from Community Source at: <https://cs.opensource.ibm.com/projects/viewProjectDetails.do?projectId=58129>. You will need a Community Source account to download the project files. Each release includes details on the project contents and details regarding how to incorporate ADF into an existing/new project. In addition to getting a Community Source account you should also join the Tivoli Endpoint Manager ADF project to among other things make sure you get notifications about future releases.

Localization

Xlat preprocessor tags are the mechanism to use for localization of all strings in an application: "Xlat tags for localization" on page 2-5. Depending on how you propagate your application files (i.e. via BDE or a build server) there are tools available for generating an Xlat file containing all localized strings for translation and translated versions of the Xlat file can subsequently be included with the propagated files to enable localization.

Left/Domain Navigation

The left navigation elements in the Console are generated via a domain specification XML file.

Dashboard Datastore

Dashboards and wizards can save data into the database using API functions, and retrieve that data using an API function (that ultimately makes use of a datastore variable session inspector). Each dashboard/wizard has its own namespace, so a wizard can use common names (e.g. "Settings") without worrying about name collisions with another wizard. In addition, variables can be flagged as "private" in which case they are only visible to a particular user. So a private "Settings" variable would store the settings for a particular wizard for a particular user. A shared "Settings" variable would store the settings for a particular wizard and would be shared by all users of that wizard, but would not be the same as a shared "Settings" variable in a different wizard.



If wizards need to share data with each other, they can. The dashboard id can be specified in both the API function for writing variable data and in the function or session inspectors used to access that data.

Key Considerations

The particular way in which TEM distributes, loads and displays applications has an impact on application development. The following are the main issues to keep in mind:

Working with Nested Folders

Prior to TEM 8.2, nested folders could not be propagated as part of a site, meaning that all of a dashboard's files needed to exist in a single flat directory. From 8.2 onward, nested folder can be propagated but there is a known issue when air gap is used. This makes nested folder a less preferred option compared to a flat folder structure.

There are two approaches that you can use nested folders:

- The `tem.library` JavaScript classes included in the API (see "Library Install") show a standard alternative approach by extracting a .zip file's contents to an external directory and loading the files from there. In the sample Dojo project included with ADF, this approach is used to load and install a full Dojo build.
- Since version 2.3.0 of ADF, a build script from `requires library` is used to combine all required javascript and other content into a single js file so that it can be propagated. Developers can still keep their source code files under nested folder for development and release the built version to production.

How IE Runs Differently in the Console

The contents of the HTML tag of an .ojo XML file get included within the body of a Console-defined HTML page. Regardless of whether the .ojo HTML contents themselves contain a body or head tag the entire contents of the .ojo's HTML are themselves included within the body tag of a generic web page, which itself loads some general JavaScript and CSS files. The content of these files can be viewed in the Reference folder in the Console install directory, additionally you can view source as you would in IE by right clicking on a Dashboard/Wizard window (to enable view source you must have already enabled the Console debug menu as described in "Loading a Project Into the Console" on page 1-3).

Additionally, IE runs as a browser control in the TEM Console and the way IE is configured is slightly different than how IE is configured to run by default outside the Console (note that this is not applicable to web reports). In the TEM Console:

- **ActiveX is fully enabled.** - Many API functions make use of the fact that ActiveX is fully enabled for IE in the Console without user prompting. Note that ActiveX is specifically fully enabled for production site content. Since TEM 8.1 custom site dashboards and wizards have been enabled and ActiveX is enabled for those as well.
- **URL is always "about:blank"** - Dashboards and wizards are always loaded with a url of "about:blank". This has an impact for example with Dojo where Dojo expects all local files to be loaded with a location protocol of "file:". This is one of the issues addressed by the Dojo support component of the ADF (see "Dojo Patch" in Part Three).



- **Setting a dashboard's DOCTYPE was enabled in 8.2** - Prior to 8.2 there was no way to set an application's DOCTYPE (it defaulted to no DOCTYPE). In 8.2 the DocType attribute was added to the OJO's Page element.

It's possible values are: HTML5, HTML 4.01 Strict, HTML 4.01 Transitional, HTML 4.01 Frameset, XHTML 1.0 Strict, XHTML 1.0 Transitional, XHTML 1.0 Frameset, XHTML 1.

If not specified it defaults to no DocType. An OJO with this attribute will result in errors on pre 8.2 Consoles, so care must be given to make sure an alternate OJO is available if an application needs to support pre-8.2.

```
<?xml version="1.0"?  
<BES>  
  <Wizard>  
...  
  <Page Name="Page1" DocType="HTML 4.01 Transitional" >
```

- **Target IE version** - Regardless of what version of IE you have installed, by default IE is run in IE 7 compatibility mode. Since IEM console version 9, this setting can be controlled by the dashboard by adding a meta tag in the Head element. The following tag sets target IE version as IE 9. The tag is not backward compatible with previous console versions, which cause the console to throw parsing error when the corresponding dashboard is opened.

```
<Head><![CDATA[<meta http-equiv="X-UA-Compatible"  
content="IE=9">]]></Head>
```

Single Xlat File

The console uses a single Xlat file for an entire site, for example a site will contain a single CHS.xlat file that holds all the strings for that locale that get referenced through Xlat preprocessor tags for all dashboards in that site. One implication of this is that since there are a number of strings used by the ADF itself and these strings have already been translated, you should make sure that the ADF translated strings get merged into your project's Xlat file. The strings will get included anyway by the same process that scans your project files, but to avoid re-translating the same set of strings the pre-translated strings should be used.

Xlat tags Only Work in Top-level files

If you are working with nested folders, note that only files in the top level will have their preprocessor Xlat tags processed. It is for this reason that localized strings for all libraries used by the ADF sample project refer to the top level resources.js file.

Sharing Data Across Dashboards/Wizards

Each item loaded by clicking on the left domain navigation is launched in a separate IE instance. This means that the DOM and variables set while viewing one page are not visible to other pages in a domain just as if the pages were loaded in separate tabs in IE (the `tem.consoleUtils.openWizard()` API function does allow DOM access for a wizard that opens another wizard but this is not applicable for a wizard opened from domain navigation). Data can



only be shared across pages by using something external to the dashboard application itself such as the Console datastore.

Relevance Results

There are certain constraints in the way that data gets retrieved via Relevance that need to be kept in mind when developing strategies for retrieving and loading data in applications:

- **Relevance always returns entire set of results at once.** When crafting a relevance expression, consider that the evaluation returns all results to the Console environment (there is no concept of a cursor, or any way to return n results at a time in relevance itself). This eliminates the possibility of a more traditional approach where the server initially sends a small subset of results, then sends more results on demand. If the developer crafts a relevance expression that returns (for example) 100,000 strings, the evaluation returns all results (in a single array in JavaScript) or no results. Therefore, allow time to craft relevance expressions with care so as only retrieve what is required by the application.

When tuning relevance queries, optimize the result set by optimizing the query itself.

- **Default to using the special RelevanceResult type returned by `tem.evalRel` as much as possible.** The custom RelevanceResult offers significant performance benefits, refer to the API documentation for `tem.evalRel()` for details. The custom RelevanceResultsStore Dojo datastore object was created for this purpose for Dojo UI components (see [“RelevanceResultsStore” in Part Three](#)).

Application Display/Layout

The Console has its own back/forward buttons that represent pages opened by clicking on the Console's left navigation. Clicking on links within a dashboard or wizard is not reflected in the Console's history, so care should be taken not to give the impression of doing a full page refresh when clicking within the application itself.

File Size/Number Propagated

In addition to keeping the file size to a minimum for optimal application load time, it is important to keep the number of files being propagated to a minimum as well.

Using requireJS and build tools

Since ADF 2.3.0, the dashboard makes use of requireJS and its custom plugins to allow developer to include ADF without worrying about its dependency. Another benefit of using requirejs is its build script that merges dependent js files into the main js file.

Run the build script

The build script can be run by running **build.bat** under **tools** folder, or running the provided



default ant task under **build.xml**.

After building, multiple files (including js, css and text files) can be merged into a single js file which is called a script layer. The sample config file **config.js** provided under tools folder defines the following layers:

- A common layer that is shared by multiple dashboards. This includes common libraries such as ADF and ADF UI components.
- A layer for each dashboard. A dashboard would include the common layer and this layer.

Difference between dev and built versions

You can load both the dev version and the built version into console for debugging. There is still some difference between the two versions:

- Files under nested folders can not use xlat tags. If you are using xlat tags in nested js files, they are not translated when the dev version is loaded. The built version brings the content up to the top level folder, thus the content is translated. If you want to see the consistency between dev and built version, it is recommended to put all localization strings in a resource file under the top level folder, and include it as a dependency.
- Loading text files using requirejs text plugin also removes xlat tags and leaves the content in English when the dev version is loaded. The built version keeps xlat tags as normal. This means, you can not test other languages using the dev version, use built version instead.
- If css file is loaded by requireJS css plugin, all images referenced in the css file are moved to the top level folder after build, and the css file is updated accordingly. This does not apply to the css files added using html link tag.

RequireJS config

RequireJS configuration is written under src/**common.js** file. You may need to modify this file to add path to new js libraries under “paths “ section, or provide dependency information for legacy js files under “shim” section. Some ADF dependency is also declared under this config file.



API

The ADF includes a JavaScript API that provides the bulk of functionality available to an application. Among the typical things you would like to do in an application that can be done using the API include:

- **Making Session Relevance calls-** the `tem.evalRel()` function can be used to make Relevance calls.
- **Creating custom content-** `tem.content.createContent()` is used for creating Fixlets, Tasks etc.
- **Storing data-** `tem.dataStore` functions allow you to work with persistent data for individual dashboards.
- **Running an external executable-** `tem.shell` functions enable running external executables via ActiveX.
- **Installing library-** `tem.library` functions enable the installation and use of libraries with nested folder structure, i.e. Dojo

JavaScript API Files

File Name	Description
ADFAPI.js	JavaScript file that contains all the API classes and functions..
json2.js	Third party library required if specifying a <code>tem.model.WizardSource</code> when using <code>tem.content.createContent()</code>

JavaScript API Overview

The following are the top-level categories of the API and a brief description of some their more significant functionality.



NOTE: Included with each release is a set of HTML-based javadoc style API documentation that provides in-depth detail regarding all individual functions and classes in the API, their parameters and example code.

tem

The root namespace contains one function `tem.evalRel()` an alias of `tem.relevance.evaluate()` that makes Relevance queries.

tem.config

This namespace contains fields and methods related to the configuration of an individual dashboard/wizard application. It includes the following:

- Methods related to setting the logging level for an application
- A way to define the set of sites that dashboard content and relevance functions will be working with.

tem.consoleUtils

This namespace contains utility functions for interaction with the Console UI and general Console functionality, including:

- Methods related to listening for console data store events and assigning refresh handler.
- Methods related to creating console filters.

tem.content

Contains functions for working with content (fixlets/tasks/analyses/baselines/computergroups) , including

- Methods related to creating and editing content.
- Methods for triggering content actions
- Methods for opening content document windows
- Methods for adding files to custom sites and mailbox sites (available in 9.0 Gilman)

tem.context

Contains functions for determining information about the context a dashboard/wizard is running in, including:Methods related to creating and editing content.

- Console version
- Preferred language
- Console vs. web report context

tem.dataStore



Contains functions for storing/deleting data in the dashboard data store. This is the primary way for individual dashboards and wizards to store their own persistent data.

tem.file

Contains classes and functions that provide interaction with the file system, including Console version

- tem.file.FileSystem works via ActiveX for a full range of functions that can operate on entire file system.
- tem.file.scratchPad for cases where ActiveX is not available, works within a Console file system sandbox.
- Methods for uploading/downloading files.

tem.javaScriptUtils

General JavaScript language support functions available for any dashboard development.

tem.l10n

Contains utility functions related to localization.

tem.library

Contains classes that provide ways to install library files (in particular library files with a nested folder file structure as nested folders in sites are not supported in pre 8.2 setups).

tem.model

Model object classes. Includes an XML serialization component that allows these model objects to be easily translatable to XML for importing into the Console for content creation/editing.

tem.relevance

Utility functions for making/working with Relevance calls.

tem.relevance.queries

Contains wrappers for generic Relevance queries.

tem.shell

Contains utility functions for executing Windows shell scripts from a dashboard or wizard via ActiveX.

Non-JavaScript Hooks



Relevance Tag

In addition to the `tem.evalRel()` JavaScript function, the relevance preprocessor tag can be used to evaluate relevance. The relevance tag can be used in the application's HTML and gets evaluated when the application's HTML is first loaded:

```
<?relevance expression ?>
```

where expression is a placeholder for a relevance expression.

Though not used as frequently as `tem.evalRel()`, this tag can be useful in cases where you need some relevance to be evaluated only once when the application first loads.

NOTE: The result is processed through the HTML inspector type, protecting the structure of surrounding HTML by escaping string results.

The link: Protocol and Opening Content Windows

Several API functions that open document windows in the Console (i.e. `tem.content.openFixlet()` and `tem.content.openAction()`) make use of the `link: protocol` behind the scenes.

When used as part of an href the `link: protocol` tells the Console to open a document window. For example clicking on the following link in the Console would result in the opening of a new Fixlet window:

```
<A href="linkid:openfixlet(0,0)"><b>Click Here</b></A>
```

Xlat tags for localization

Xlat preprocessor tags are used for localization of dashboard and wizards. There are three different tags based on what context the translation occurs in, since different contexts have different escaping rules:

- `hxlat` -- used inside HTML (inside an ojo file or fixlet HTML)
- `jxlat` -- used inside javascript string literals (inside `<script>` tags in an ojo file or fixlet HTML, or inside a js file)
- `rxlat` -- used inside relevance string literals and used in conjunction with the `tem.l10n.format()` function.

The following examples illustrate their usage, how special characters are escaped in different contexts, and how the translations are performed.

hxlat in HTML:

```
<html>
  <body>
    <?hxlat >>Hi "famous" bearmo<<?>
  </body>
</html>
```

Outputs as:



```
<html>
  <body>
    &gt;&gt;Hola osomo "famosomo"&lt;&lt;;
  </body>
</html>
```

jxlat in JavaScript:

```
<script>
  var xlat = "<?jxlat >>Hi "famous" bearmo<<?>";
</script>
```

Outputs as:

```
<script>
  var xlat = ">>Hola osomo \"famosomo\"<<";
</script>
```

If you want to generate a parameterized localized string you need to use rxlat and the `tem.l10n.format()` function:

```
<script>
  alert(tem.l10n.format('<?rxlat Showing {0} out of {1} fixlets ?>',
    1000, 'number of bes fixlets'));
</script>
```

Would generate an alert with something like the following string:

Showing 1,000 out of 12,000 fixlets

Additionally, you could use rxlat with the format inspector in HTML:

```
<html>
  <body>
    <?relevance format "<?rxlat There are >>{0}<< fixlets in site
    "{1}".?>" + number of fixlets of current site + display name of
    current site?>
  </body>
</html>
```

becomes:



```
<html>
  <body>
    <?relevance format "Hay >>{0}<< fixlitos en sito %22{1}%22." +
      number of fixlets of current site + display name of current site?>
  </body>
</html>
```

which becomes:

```
<html>
  <body>
    Hay &gt;&gt;5.000&lt;&lt; fixlitos en sito "BES Supportito".
  </body>
</html>
```

which ultimately gets rendered as:

```
Hay >>5.000<< fixlitos en sito "BES Supportito".
```

Namespacing

JavaScript API functions are fully namespaced, though care should be taken to include the ADFAPI.js file as early in the document as possible as actual namespacing occurs there.

Some of the API functions rely on DOM elements and there is the possibility of naming collisions. The full list of ID's used by supporting DOM elements is as follows:

TheUtilityScripts

{ActivateAnalysis elements}

```
ActivateAnalysisData, ButtonActivateAnalysis theActivateAnalysisIDList,
  theActivateAnalysisReactivateFlag,
  theActivateAnalysisDeactivateFlag, theActivateAnalysisSuccessFlag,
```

{StopAction elements}

```
StopActionData, ButtonStopAction, theStopActionIDList,
  theStopActionSuccessFlag
```

```
RegisterRefreshHandlerData, ButtonRegisterRefreshHandler,
  theRegisterRefreshElementID, theRegisterRefreshSignalName
```

{File/Folder browsing elements}



```
BrowseData, ButtonBrowse, theBrowseExtension, theBrowseFilters,  
    theBrowseForFolderFlag, theBrowseInitialPath,  
    theBrowseFileMustExistFlag  
  
theBrowsePathMustExistFlag, theBrowseNoValidateFlag,  
    theBrowseHideReadOnlyFlag, theBrowseOverwritePromptFlag,  
    theBrowseCreatePromptFlag, theBrowseNoReadOnlyReturnFlag,  
    theBrowseNoTestFileCreateFlag, theBrowseAllowMultiSelectFlag,  
    theBrowseResultCancelFlag, theBrowseResultFullPath  
  
{LoadPresentation elements}  
  
LoadPresentationData, ButtonLoadPresentation, thePresentationPath,  
    thePresentationResult  
  
{ScratchPad elements}  
  
ScratchFileOpsData, ButtonScratchFileOps, theScratchFileOperation,  
    theScratchFileInputPath, theScratchFileOutputPath,  
    theScratchFileRecurseFlag, theScratchFileResult  
  
{ConnectionInfo elements}  
  
ConnectionInfoData, GetConnectionInfo, theConnectionInfoCurrentDSN,  
    theConnectionInfoCurrentUser
```



In addition to the API with its core set of functionality, the ADF provides a UI component library built off of jQuery plugins and libraries and a sample dashboard that shows how to implement a basic application.

For teams that prefer to work with the Dojo framework there is a set of "support" code that enables Dojo applications to run in the Console. A sample application is also provided that shows how to build a dashboard using a particular set of core Dojo components and a set of Tivoli custom Dojo widgets.

All libraries included with ADF and its samples have either gone through the IBM process for approval of Open Source libraries or are internally developed IBM libraries (i.e. Tivoli Extensions for Dojo).

ADF UI (jQuery-based) vs Dojo

In terms of whether to build your application's UI with the ADF's UI components vs. using Dojo UI components, it is recommended that unless your development team has extensive Dojo development experience you should use the jQuery UI components and libraries as there are fewer issues in terms of working with nested folders and the embedded browser control and it has a shallower learning curve. Dojo development is supported by the ADF, but the recommended default is jQuery. Using both together would be an option as well, but would have the same caveats as pure Dojo development. The ADF API is independent of any particular JavaScript framework, so any combination of jQuery and Dojo UI libraries (or even plain HTML/JS) is possible as the front end of an application.

Look and Feel

The overall UI strategy for applications is to converge with IBM initiatives for standardizing look and feel. To this end the custom ADF UI jQuery components are built on a custom theme that mimics the look and feel of the Tivoli Extensions for Dojo library which itself is based on the Dojo claro theme.

Using the ADF UI Components

How to Use the Sample Project

The ADF includes a sample.ojo file that shows how to set up a dashboard using the ADF API with ADF UI components which are based on several jQuery plugins and libraries.

How to Create a New Dashboard/Wizard

The ADF includes a template.ojo file that can serve as the basis for new applications. Note that it does not contain minified versions of library files which should be swapped in before the final propagation to production.

Critical Issues



- Though sample and template OJO's rely on the nested folder feature, it is possible to use the same `tem.library.JavaScriptLibrary()` approach as `sampleDojo.ojo` by creating a zip file of the files used by your dashboard.
- All third party library files used by the ADF UI have minified versions of their files in the `libs` folder which should be swapped in before release to production. Additionally make sure to prune away any unused third party library files such as unused plugins and controls. Note that the library files included in `sample.ojo` may include plugins you don't need for your application.
- Keep the `resources.js` file that has all the ADF UI localized strings in it in the top level of your project see "Xlat tags only work in top-level files" on page 1-8
- Use the API's `tem.logging.debug()` and `tem.logging.error()` functions as much as possible to provide insight into application's workings. These functions get initialized to write to the dashboard's log component as well as "console" if it's available. Also use `temui.showErrorDialog()` as your standard error popup.
- When debugging, keep in mind that applications in the Console are run in IE 7 Standards Mode regardless of their browser version: "How IE Runs Differently in the Console" on page 1-6.

UI Components/temui

The following are the UI components currently available in the ADF. For more detailed information regarding usage please refer to the API docs and sample project. For the contents of the third party libraries that these UI components are built on including documentation and sample code, please refer to the `libs` folder in the ADF release.

NOTE: Included with each release is a set of HTML-based javadoc style API documentation that provides in-depth detail regarding all individual functions and classes used for the UI components, their parameters and example code.

Charts

Basic charting components are based on the jqPlot library: <http://www.jqplot.com>

- **barChart()**
- **columnChart()**
- **pieChart()**
- **chartContainer()**

DataGrid

- **dataGrid()**- Based on the SlickGrid dataGrid library: <http://github.com/mleibman/SlickGrid>
- **gridContainer()**



Health Checks Dashboard.

- `healthCheck()`
- `healthCheckGroup()`

General UI Widgets

- `temHeader()`
- `temTopNav()`
- `filterInput()`
- `showErrorDialog()`
- `showLogDialog()`
- `analysisWarning()`
- `message()`
- `popupMessage()`
- A number of widgets are built on top of the jQuery UI library: <http://jqueryui.com>. There are a number of general components in the library (i.e. dialog, slider) that are available to use as well.

Form Validation

jQuery plugin **temValidate()** is used to add validation to form inputs. The plugin is based on jQuery validate plugin (<http://docs.jquery.com/Plugins/Validation>) with customization of error message display and additional validators:

- `data-tem-validate-relevance`
- `data-tem-validate-function`

Layout

While it doesn't contain any actual UI components, YUI's CSS Grids is a stylesheet that is used in a number of ADF components and can be extremely helpful in general page layout: <http://yuilib.com/yui/docs/cssgrids>








Use of colors

Status Color

When color is used to indicate the severity of an item, use the following colors:

Meaning	Color	Hex
---------	-------	-----



Critical		#e51717
Warning		#ffb31a
Minor		#ffff1a
Nominal		#33cc14
Informational		#2ea8e5
Unknown		#a6a6a6
Undefined		#6963a6

Internal Localization

Localization for the ADF UI components is all managed through the Xlat mechanism and the resources.js file except for the jQuery UI datepicker component. Refer to the datapicker component's documentation for details.

Default Application Initialization

By default every application will get initialized in the following manner:

- Header and top navigation get stylized.
- A global error handler is assigned to window.onerror that displays the standard error dialog UI component.
- A key listener is added so that the Ctrl-Shift-D key combination brings up the log settings dialog.
- It will default to the user's last selected preference regarding logging level.
- tem.logging.debug() and tem.logging.error() get set up to write to logging UI component as well as "console" if it's available.

Support for AngularJS

AngularJS support in console

Since ADF version 2.3.0, a sample dashboard that makes use of AngularJS library is provided in the package. In general, angularJS can be used in the console environment with the following restrictions:

- If the dashboard is not targeting specific IE version, the default version is IE7 compatibility mode. Because of this, angular directives can not be used as element name, use them as attribute name instead.
- ng-include doesn't work with external template file because of local file access restriction. Consider using requirejs to load the file content and use it in directive template or cache it using \$templateCache service.



- Errors and warnings thrown in angular components are captured and written to debug console. To see these errors and warning, bring up the firebug console or check Ctrl-Shift-D logging dialog.

AngularJS localization setting is set to the console preferred language when ADF angular components are added into the dashboard.

ADF AngularJS components

ADF version 2.3.0 started adding AngularJS support for some UI components. These components are defined under “adf-ng” module. The module can be loaded using requirejs.

The following components are added in 2.3.0:

- adf-top-navigation
- adf-button
- adf-chart-container
- adf-bar-chart, adf-column-chart, adf-pie-chart
- adf-tab-set, adf-tab

Support for Dojo Dashboards

How to Use the Dojo Sample Project

sampleDojo.ojo shows how to use the ADF API with a set of UI components that are based on Dojo core components and a custom library developed by a Tivoli team: Tivoli Extension for Dojo (TED). There is also support code for Dojo that provides standard workarounds for some of the quirks that directly affect how Dojo works when run in the Console’s browser control (see “[Key Considerations](#)” in [Part One](#)). The Dojo sample also shows how to work with the API’s `tem.library.JavaScriptLibrary` functions to install a set of library files when nested folders are not an option (it essentially unzips the contents of a zip folder to an external temp folder).

Please note that the TED library included with the Dojo sample has since moved to be part of the Tivoli Widget Library. Feel free to use whatever IBM Dojo library makes the most sense for your project, the API part of the ADF is independent of any particular JavaScript framework, the TED library is included with the sample project mainly to show what’s possible.

Critical Issues

- Consider using the Dojo build system as an alternative for nested folders as well as a way to minimize the number and size of propagated files. <http://dojotoolkit.org/reference-guide/1.8/build/>
- Keep the `resourcesDojoUI.js` file that has all the ADF UI localized strings in it in the top level of your project see “[Xlat tags only work in top-level files](#)” in [Part One](#)
- Patching Dojo with an external file is not possible in all versions of Dojo, you may need to modify the source code directly: “[Dojo Patch](#)” in [Part Three](#)



Dojo Patch

NOTE: Patching Dojo is known to work with the version of Dojo included with sampleDojo.ojo. It is known not to work in Dojo 1.7.2 where the source code would need to be modified directly

Included in the Dojo support module is a file called ADFDojoPatch.js. This file overrides certain settings and functions in the Dojo source code to accommodate some of the differences with how the Console uses IE to load the dashboard's HTML as opposed to how a typical web application would get loaded into IE as a standalone browser (["How IE Runs Differently in the Console" in Part One](#)), specifically:

- It accounts for a location protocol of "about:" in the same way it accounts for the "file:" protocol.

Additionally, it ensures the use of the Msxml2.XMLHTTP.6.0 ActiveX component for making XMLHttpRequests. This component gets installed with the Console if it doesn't exist on the system and has been tested to work for making XMLHttpRequests from within the Console.

NOTE: When using isDebug=true in your djConfig settings for Dojo, errors will occur as Dojo is trying to make an XMLHttpRequest call before the patch overriding code has been loaded. In this case you will need to make the modifications in the patch file directly in the Dojo source itself.

IFrame Alternative

As an alternative to using ADFDojoPatch.js, you can load your dashboard HTML through an iframe. This works around the about:blank and XMLHttpRequest issues addressed by the patch file. There are several known drawbacks with this approach however:

- Copy/paste does not work with UI elements that get displayed in the iframe.
- Some of the backend functionality that relies on triggering DOM events does not work as well (including tem.content.activateAnalyses(), tem.content.stopActions(), tem.consoleUtils.registerRefreshHandler() and file browsing and tem.file.scratchPad methods).
- The code in wizards.js file has to be loaded separately to initialize the hooks properly.

For these reasons it is recommended to use the patch approach.

Library Install

Patching DojoAmong other sample code, the sampleDojo project includes an example of how to do a local Dojo install for use with a particular dashboard:

File Name	Description
-----------	-------------



BFArchive.exe,
dojo_1.5.1_WTED
_1100.besarchive

File compression utility and file archive containing TED 1.1 and Dojo 1.5 components. These files get installed as the library for the sample project.

```
var dojoLibrary= new tem.library.JavaScriptLibrary('dojoWTED',  
'dojo_1.5.1_WTED_1100', 'dojo_1.5.1_WTED_1100.besarchive');  
dojoLibrary.install();  
dojoLibrary.loadJavaScript("dojo/dojo.js");  
dojoLibrary.loadCSS("dojo/resources/dojo.css");
```

Internal Localization

The sample application includes a piece where it specifies a user's locale for Dojo to use with its own internal localization mechanism:

```
var djConfig = {  
  parseOnLoad: true,{  
    locale:tem.context.getUserLocale()  
  };  
};
```

Dojo uses the specified locale for localizing the strings defined internally in the framework. The `tem.context.getUserLocale()` utility function returns the user locale (for details on how this locale is determined refer to the API docs.)

RelevanceResultsStore

The sample project contains an example implementation of a `RelevanceResultsStore`. This is an implementation of a Dojo datastore that provides a read-only wrapper for the special JS type that gets returned from `tem.evalRel()`. This datastore encapsulates an optimal way for working with a simple set of Relevance results. For UI and Relevance results that fit its criteria it can provide significant performance benefits in terms of data loading times and UI responsiveness.

Default Application Initialization

The Dojo sample has some of the default application initialization as the standard sample:

- A global error handler is assigned to `window.onerror`
- A key listener is added so that the Ctrl-Shift-D key combination brings up the log settings dialog.
- `tem.logging.debug()` and `tem.logging.error()` get set up to write to logging UI component.



Part Four

Notices

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs



(including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
2Z4A/101
11400 Burnet Road
Austin, TX 78758 U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

TRADEMARKS:

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at <http://www.ibm.com/legal/copytrade.shtml>.

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

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.



Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.