Introduction to the IBM Case Manager JavaScript API

Type of Submission: Article
Title: Introduction to the IBM Case Manager JavaScript API

Keywords: IBM Case Manager Client, JavaScript API, Design pattern

Author: Mayes, Lauren
Job Title: STSM, Product architect, Case Manager and Enterprise Content Management
Email: lmayes@us.ibm.com
Bio: Chief architect of IBM Case Manager.
Company: IBM

Author: Tian, Xiao Ji
Job Title: Advisory Software Engineer, Case Manager and Enterprise Content Management
Email: tianxj@cn.ibm.com
Bio: Technical lead of the IBM Case Manager Client.
Company: IBM

Author: Guo, Ming Liang
Job Title: Senior Software Engineer, Case Manager and Enterprise Content Management
Email: guoml@cn.ibm.com
Bio: Architect of the IBM Case Manager Client.
Company: IBM

Author: Shenoy, Anuradha S.
Job Title: Lead developer, IBM Case Manager Client
Email: shaenoya@us.ibm.com
Bio: Lead developer of the IBM Case Manager Client.
Company: IBM

Author: Gao, Ying Ming
Job Title: Software developer, IBM Case Manager Client
Email: guoyingm@cn.ibm.com
Bio: Developer of the IBM Case Manager Client action framework.
Company: IBM

Abstract: IBM® Case Manager includes a JavaScript API that you can use to customize your case management client application. This article provides an overview of the client architecture and the JavaScript API, and describes the ways you can customize your client application.

Overview

Aligned with the IBM Enterprise Content Management client strategy, IBM Case Manager Client is built on top of IBM Content Navigator. They share a similar architecture design and consistent user experience. IBM Case Manager Client reuses some visual and model toolkits from IBM Content Navigator.

IBM Case Manager Client is built around the core ideas of reusable page widgets and customizable pages.

IBM Case Manager Client includes a set of IBM Case Manager page widgets, each of which are designed to provide a particular case-related function. Customers can also build their own page widgets. A page contains several page widgets. Typically, the widget functions do not overlap, but work together through event notifications to present a cohesive user interface.

The whole process of building a case solution is divided into design time and runtime:

- At design time, with the Page Designer, a Business Analyst can create a solution page by dragging and dropping the IBM Case Manager page widget into the page, configuring settings, and adding event wiring between the page widgets. Each solution role can be associated with a set of different pages.
- At runtime, after the solution is deployed, users can select the solution and a role to work with. The IBM Case Manager Client will download and render the set of pages that was associated with the selected role in IBM Case Manager Builder.

For a particular case solution, the page widgets provided by IBM Case Manager usually cannot meet all solution requirements. With the IBM Case Manager Client JavaScript Toolkit API, developers can build their owner page widgets or customize IBM Case Manager page widgets to meet particular requirements. After packaging and registering the custom page widgets, they are available in both design time and runtime.

Design goals

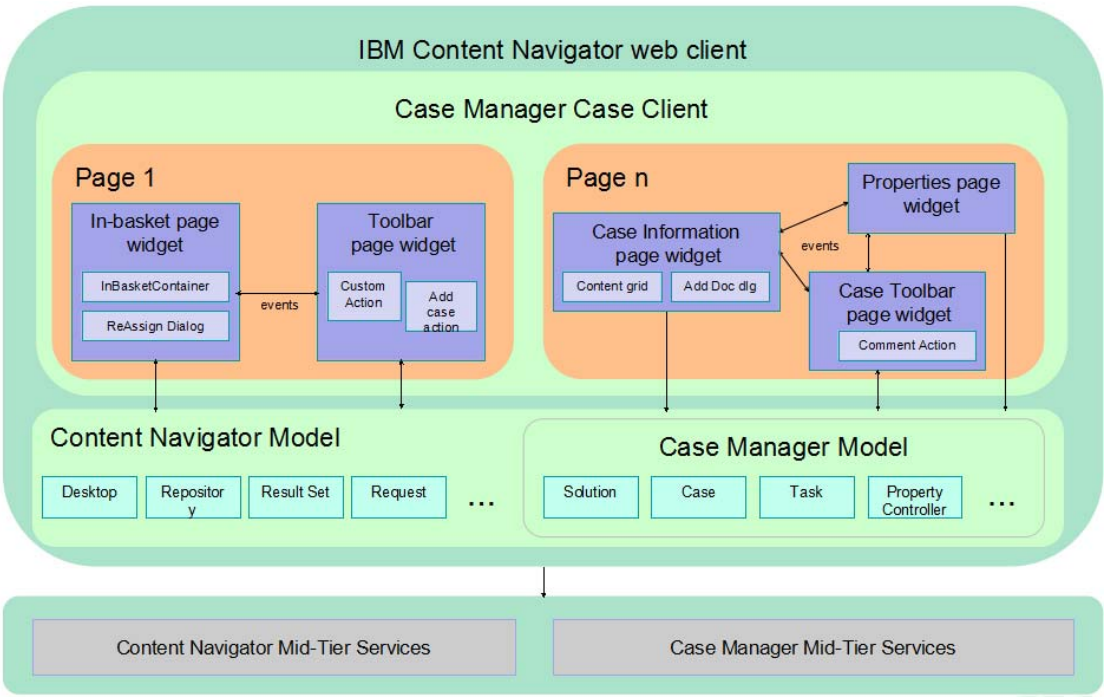
The primary goal is to provide a highly customizable Case Manager Client with reusable page widgets. Customization has two aspects:

- A Business Analyst can customize a page with a different set of page widgets, and customize the widget behavior by using the edit settings and event wiring.
- A developer can build a custom page widget based on the IBM Case Manager JavaScript Toolkit APIs.

IBM Case Manager Client encourages the following design principles:

- Build IBM Case Manager visual components that can be wired together to do larger functions without coding. For example, if the work page is used for browsing and processing work items, there is usually only an in-basket page widget. A Business Analyst might add a case information page widget to the page, and wire the in-basket 'icm.SelectWorkitem' event with the case information 'icm.SendWorkItem' event handler. When a user selects a work item in the in-basket, the case information shows the information of its parent case.
- Define highly granular page widgets to make it easier to replace parts without affecting other parts of the whole function. For example, the cases page is used for searching cases. If there is additional information, such as highlighting, that needs to be shown in the case list, a developer might build another case list to replace the old one. But, the search and case information is still used in the page, and the whole case search function is not broken.
- Build pages and page widgets that are portable to custom applications.
- Do not try to reinvent wheel. Reuse the toolkits and infrastructures from IBM Content Navigator as much as possible.
- Be consistent with IBM Content Navigator design principles.

Client architecture



The **IBM Case Manager model** is a set of JavaScript toolkit APIs for create, read, update, and delete (CRUD) data operations against server-side IBM Case Manager

domain objects, such as solutions, cases, and tasks. It is well organized, and positioned as the case domain language's JavaScript implementation. The model simplifies client-side programming against a case domain. The model is publically available for customers to build their own page widgets for case management. IBM Case Manager page widgets use this model as the model layer in the model-view-controller (MVC) architecture.

IBM Case Manager widgets are a set of pure JavaScript visual toolkits, which are the view layer in the IBM Case Manager page widget MVC architecture.

The **Action** framework provides the general mechanism for a page widget to add actions to its toolbar and context menu. It consists of actions, action manager, toolbar, contextual menu, and page designer action configuration. Action is a small business operation unit based on context resources. Both the toolbar and contextual menu are dojo widgets that can be represented as a toolbar or menu in any page widget. Page designer action configuration is a page designer function that provides the ability to configure appropriate actions for a specific toolbar or menu in a page widget. The actions set for a toolbar or menu are rendered as a series of buttons or menu items shown in a toolbar or contextual menu. Action Manager is responsible for controlling the status of the buttons, menu items, toolbars, and contextual menus.

The **Event** mechanism provides the means for unidirectional communication from one page widget to another page widget.

The **Coordination** mechanism provides the means for bidirectional communication between page widgets.

An **IBM Case Manager page widget** is a particular visual component that contains both user interface (UI) and business logic. The page widget, which relies on IBM Content Navigator and the IBM Case Manager model layer for data CRUD operations, is a classic MVC unit with an event interface. Usually, the page widget exposes configurable preferences for a Business Analyst to customize. Page widgets are designed to be reusable. They are usually self-contained and have a single responsibility.

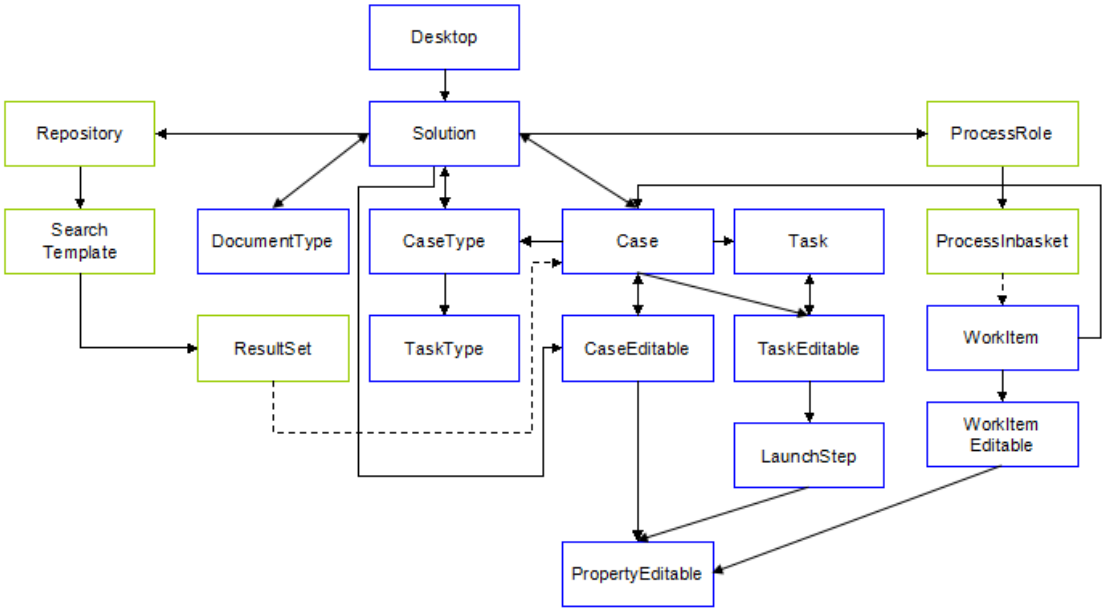
An **IBM Case Manager page** is visual component composed of several page widgets that work together to fulfill a larger function. The page controls the layout of the page widgets, sets up the event wiring between page widgets, and serves as an event broker to route events from the publisher to the handler. Unlike IBM Content Navigator visual components, a page is no longer an MVC unit. There is no centralized controller between page widgets; they work together by means of unidirectional communication (event) and bidirectional communication (coordination). In a page, if a page widget does not fit a custom solution, it can be replaced by a custom page widget.

The **IBM Case Manager container widget** is the content pane of the case feature. It provides a selector for the user to select a solution and a role. The container widget shows the pages that are associated with the role in a tab container. By default, there are cases and work pages. Users can start working from these pages and launch case and work items in new pages for further processing.

Page designer is a tool in Case Manager Builder that a Business Analyst can use to create a page and define the page layout, drag and drop page widgets into the layout, configure the page widget preferences, and add event wiring between page widgets. Page designer dumps the page design as a physical dojo Dijit.

Property view designer is a tool in Case Manager Builder that a Business Analyst can use to create a property view, define the view layout by embedding layout controls like a list, table, or tab, and drag and drop properties into layouts. The defined property view is used by the property page widget to show case properties.

IBM Case Manager Model API



The blue boxes in this diagram show the IBM Case Manager Model Object Navigation. The green boxes represent IBM Content Navigator Model Objects.

IBM Case Manager Model APIs share a similar design with IBM Content Navigator Model APIs, and they share same base classes.

Each IBM Case Manager model class represents a case domain model object, which has functions defined for domain object creation and for retrieving and updating properties. Also, there are functions defined on each model class for referencing other IBM Case Manager model objects that are related to it. Page widgets might need information to become workable, and page widgets can **navigate** through IBM Case Manager model objects to get context that is needed for them to render content. For example, the Case Information page widget in the work detail page receives the work item model object from event payload, but it needs the case model object to render content. Thus, to minimize the pass through of model objects from one page widget to another, the Case Information page widget can navigate to get the case model object from the received work item model object.

The case domain objects are mapped to server-side Content Engine and Process Engine objects. The IBM Content Navigator model layer APIs are already capable of searching and retrieving those server-side objects, but they lack the semantics for the case context. The IBM Case Manager Client is designed to fully leverage these capabilities.

Usually, IBM Case Manager page widgets search and retrieve case domain objects by using IBM Content Navigator model APIs, then convert the objects to IBM Case Manager model objects by using static bridging APIs defined on IBM Case Manager model classes. This **IBM Content Navigator model to IBM Case Manager model bridging and delegation** design pattern is used in many cases when it is possible. But, in cases where this pattern does not fit, IBM Case Manager model objects still need to communicate with the IBM Case Manager plug-in APIs.

In many cases, page widgets in a page or across pages are mapped to same or related case domain objects. When they are in same page, any intermediate changes in one page widget must be communicated to the other page widget, which can update itself accordingly in case they share the same properties or there are constraints between their properties. When the widgets are in different pages, the page widgets in one page should be notified after the page widgets in another page commits changes. Two design patterns are used to meet the requirements.

IBM Case Manager **editable model** objects are of a scratch pad pattern design: a page widget can continually update its attributes before committing changes to server-side case domain objects. Usually, an IBM Case Manager model object has a paired IBM Case Manager editable model object, which can be created by calling IBM Case Manager model object’s createEditable() function.

Page widgets in the same page share the same IBM Case Manager editable model object, which is carried in payload by the same event. The page widgets listen on the onChange()

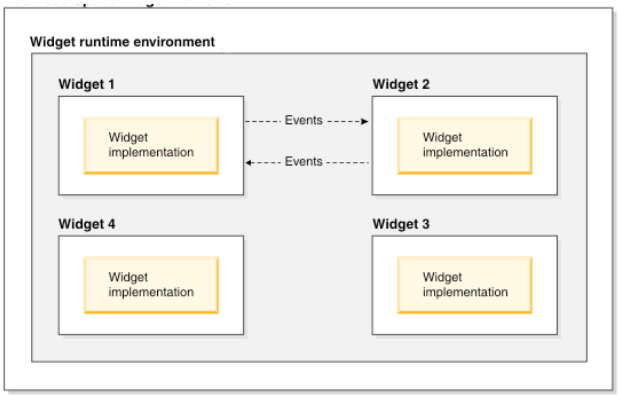
function of each attribute in the shared IBM Case Manager editable model object for notification when an update occurs.

Page widgets in different pages operate on different IBM Case Manager editable model objects even though they represent the same case domain object. These editable model objects are all connected to each other through the underlying IBM Case Manager model object. Any changes committed on one editable model triggers the refreshment of other IBM Case Manager editable objects. Page widgets can be notified by listening on the onRefresh() function of the IBM Case Manager editable object.

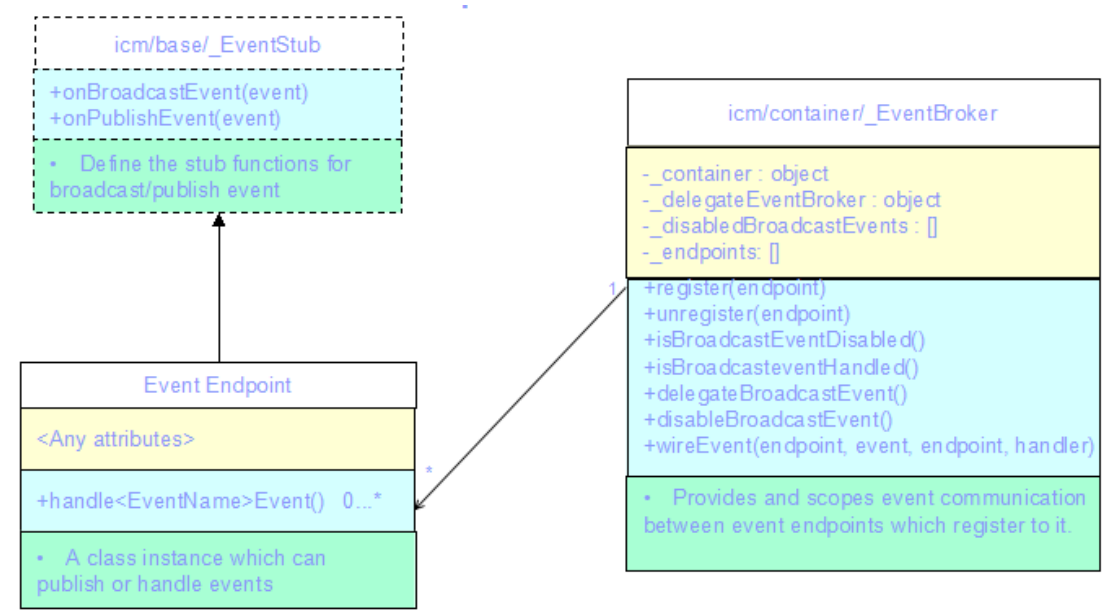
In summary, in IBM Case Manager Client, for page widgets to stay in sync with the underlying model object, the most convenient way is to hook to the model object notification interface.

Event

Page widgets communicate with other page widgets on the same page by using events. When a user does something like clicking a button or selecting something in the widget, or a when a widget receives an information update from a server, the widget can fire an event to inform other widgets on the page. The widgets that receive the event can respond in some way. For example, the responding widget could update the display to show information based on the information displayed by the firing widget, communicate with a server, or fire its own event.



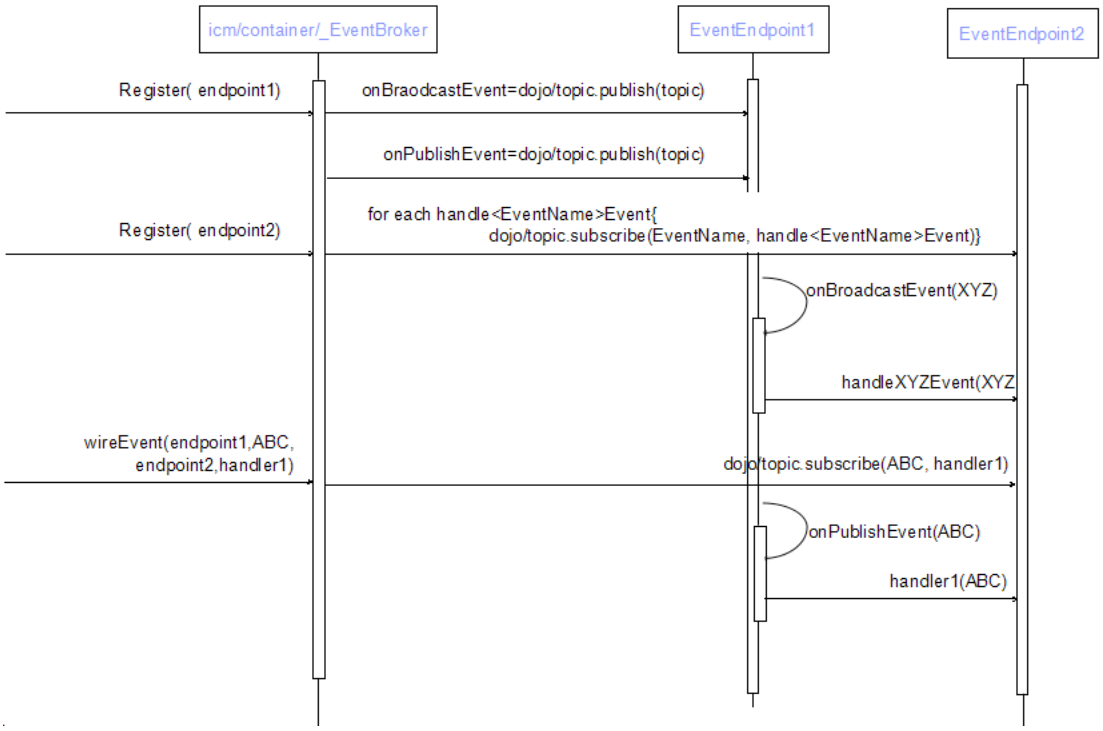
IBM Case Manager Client supports event broadcasting, event wiring, and publishing events across pages. The event mechanism is built on top of dojo/topic APIs, with additional scope for limiting the event publish/subscribe action to inside a page rather than globally.



The **Event endpoint** could be any dijit class, which wants to publish/subscribe events. The digit class must have the functions of onBroadcastEvent(), onPublishEvent(), and handle<EventName>Event(). Usually, an event endpoint is inherited from _EventStub. The two functions defined are simply stub functions for _EventBroker to hook on to.

A group of event endpoints can talk to each other if they are registered to the same **_EventBroker**. Usually, each page has an _EventBroker, and the page widgets in a page can talk through the _EventBroker. When a page widget calls the event stub functions and passes the event ID and event payload as parameters, the _EventBroker will route the event to right event handler.

The following diagram shows the sequence for endpoint register and event publish.



When you register an event endpoint to an _EventBroker, it will scan and hook up to the event stub functions of onBroadcastEvent() or onPublishEvent(), and set up publish or subscribe topics for the handle<EventName>Event function. (<EventName> can be any event name.)

When an event endpoint calls its onBroadcastEvent() function with event XYZ, the handleXYZEvent() function on another event endpoint will be called with event XYZ as a parameter. (The _EventBroker matches the event handler with naming conversion of handle<EventName>Event.)

If an event wiring was added, when the publish event broker calls its onPublishEvent() function with event XYZ, the event handler function of the subscriber endpoint will be called with the event XYZ as a parameter.

Any broadcast event will be published first.

Coordination

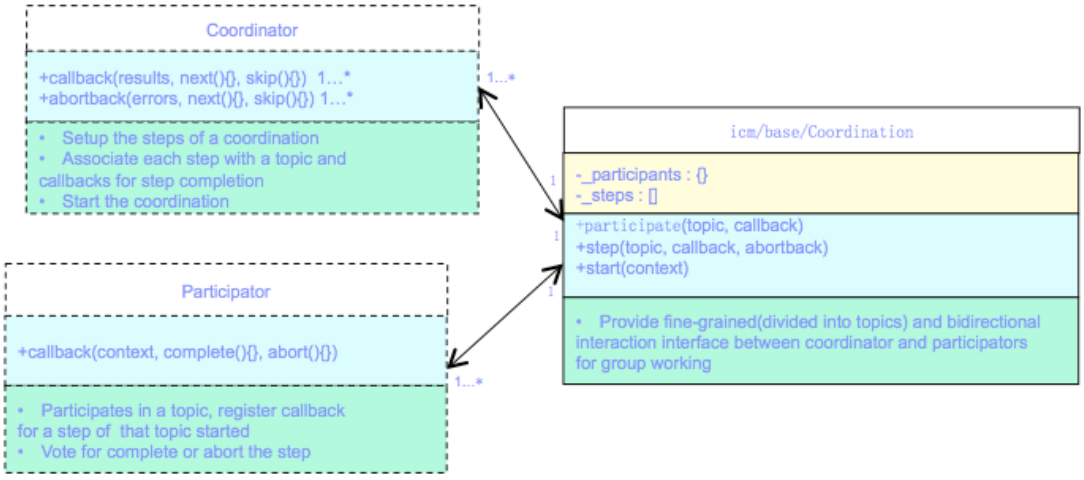
In some IBM Case Manager scenarios, it's necessary to have several page widgets coordinate to work together to fulfill a work item. For example, in a work item form page, there are three page widgets – form, attachment, and work item toolbar.

If a user clicks the Save action button in the work item toolbar, several steps of coordination are needed to fulfill the save:

1. The work item toolbar asks all page widgets to commit changes. Each page widget commits its changes, and then notifies the work item toolbar. After all page widgets complete their work, the work item toolbar moves to the data validation step.
2. The work item toolbar asks all page widgets to check if there are any validation errors. After all page widgets finish the validation, they notify the work item toolbar, and the toolbar moves to the before save step.
3. In the before save step, the form widget saves the form itself as an attachment of the work item. Then, it notifies the work item toolbar to move to the next step.
4. The next step is the save step. Here, the work item toolbar commits the changed model to the server. Next, the work item toolbar starts the after save step.
5. In the after save step, the form page widget and attachment widget clear their dirty state, and notify the work item toolbar of their completion. At this point, the coordination between page widgets for the save action is complete.

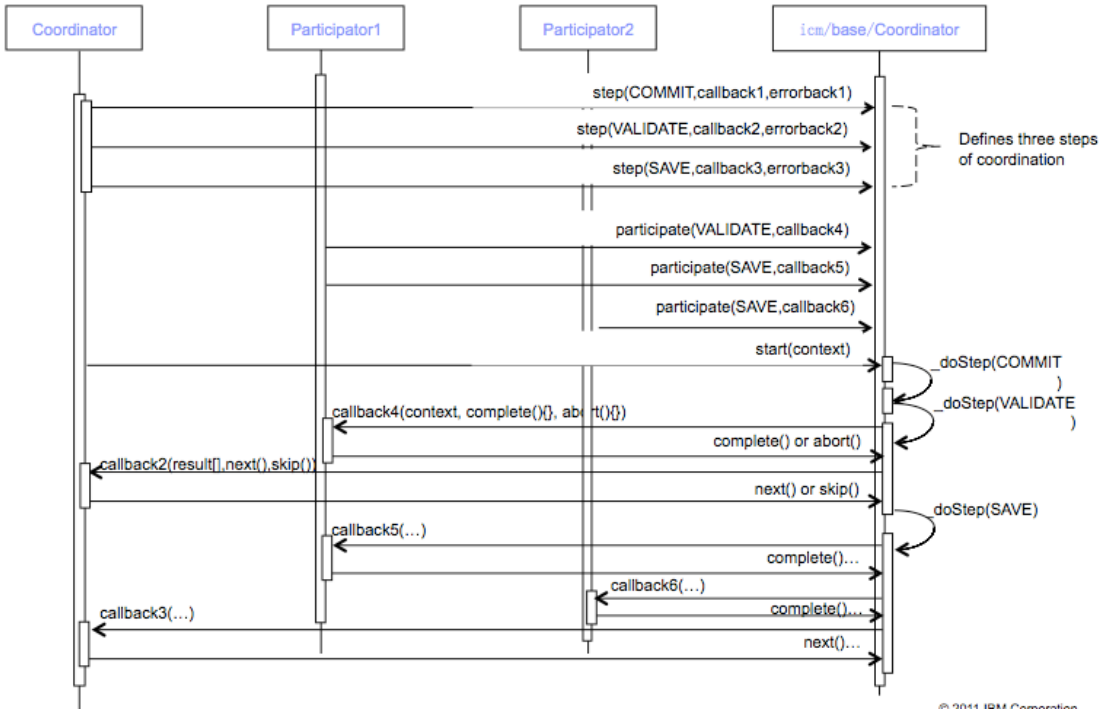
The coordination between page widgets has characteristics of:

- Bi-directional communication.
- Fine granularity of coordination steps.



The icm/base/Coordinator is a utility class for the coordination between page widgets. There are three parts consistent with coordination: one **coordinator**, multiple **participators**, and several **coordination topics**. A coordination can have several fine granularity steps. Each step associates with a coordination topic, which is a subject for coordinator and participators to join together. A coordinator can define the coordination steps and related topics, then start and control the coordination flow. A participator can

join any interested topic, work for a topic when it starts, and feed the result back to the coordinator.

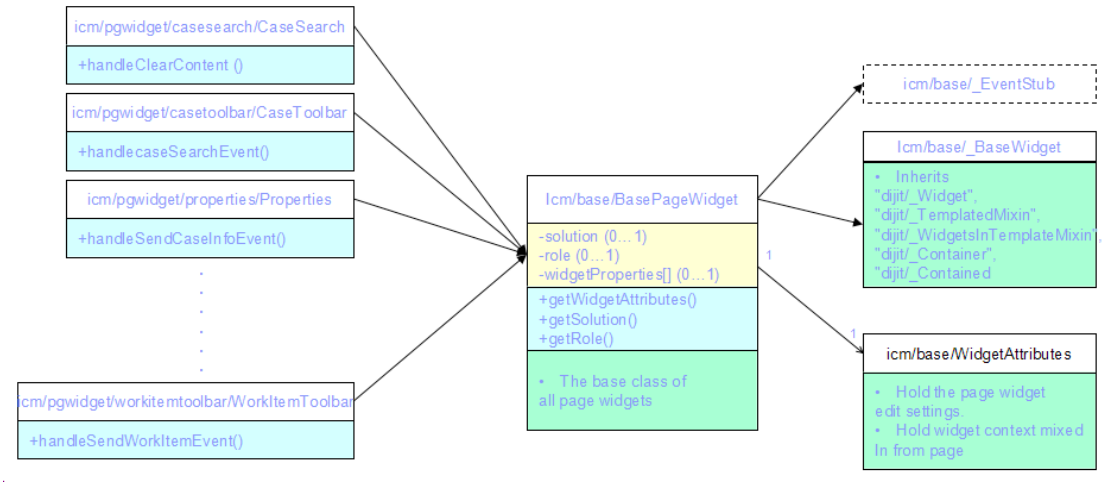


A coordinator sets up a sequence of coordination topics (such as, beforeSave, Save, and afterSave) by step(topic) function, and registers callback and errback to be called when all participators complete their work. The participators register callbacks to the interested topic. After the coordinator starts the coordination, the coordination topics will be started in an order defined by the steps.

When a coordination topic is started, every registered participator's callbacks will be called with the complete() and abort() functions passed in as parameters. They call the complete() function with the result as a parameter to tell the coordinator that they fulfilled their work, or they call the abort() function with an error as parameter to tell the coordinator that they finished their work with errors.

After each participator's callbacks are finished, the participator's register callback or errback will be called with all participators' results in an array as parameter. In the callback, two functions are passed in as parameters: next() and skip(). Depending on the results, the coordinator can determine if it wants to process the next coordination topic by calling the next() function, or skip the remaining topics by calling the skip() function.

Page widget



Basically, a page widget is a dojo dijit. A page widget is also an event endpoint and a container of actions. The Dojo dijit programming guide is the primary guide for developing a page widget. The lifecycle of page widget is typically the dojo dijit lifecycle, which connects to its parent page's life cycle (see http://dojotoolkit.org/reference-guide/1.8/dijit/_WidgetBase.html).

A page widget has preferences for configuration. In page designer, a Business Analyst can customize a page widget by configuring specific preferences, add/remove actions, and enable/disable event broadcasting. In runtime, the configuration will be mixed in to the page widget by its parent page; the page widget will behave according to the configuration settings.

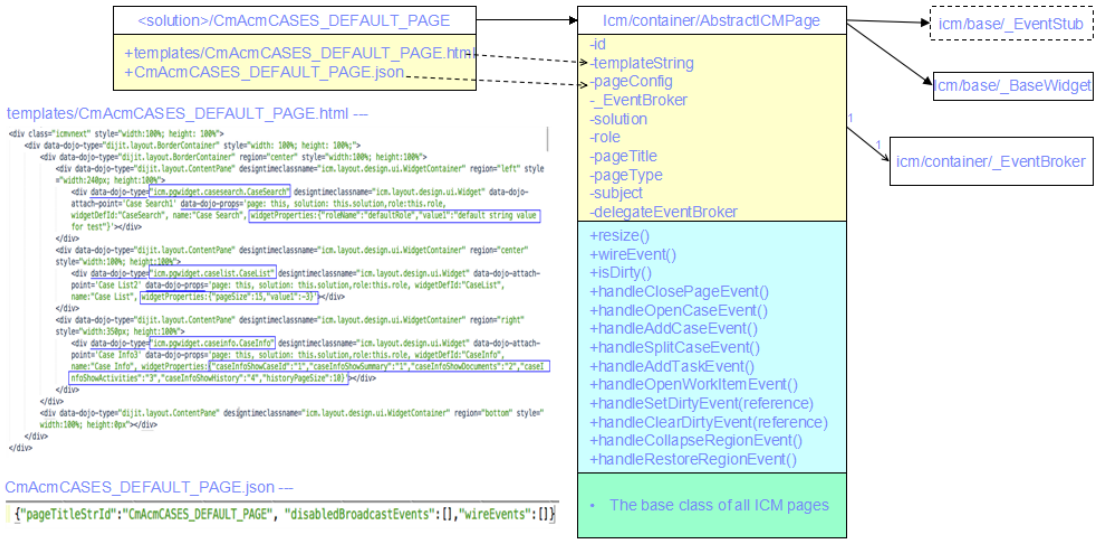
A page widget and its actions both publish events through the page widget's event stub functions. It's not necessary to distinguish whether an event is published by the page widget or its actions.

In some scenarios, page widgets in a page need to bi-directionally communicate to fulfill a function. For example, the work item toolbar tells the properties widget and attachment widget to update their changes into the shared work item editable object. After the properties and attachment widgets both communicate that their changes have been updated, the toolbar commits the changes in the editable model object to the server. The coordination object is used for this purpose. Usually, a topic is selected for coordination. One page widget (such as a toolbar) joins the topic coordinator, and other page widgets join the topic as participators (such as properties and attachments).

The page widgets inside an IBM Case Manager page can expect that the parent page will mix in the working context of the current solution and role to it.

Developers can follow the MVC pattern to build custom page widgets by leveraging existing IBM Content Navigator/IBM Case Manager model APIs and visual components. Also, many IBM Case Manager page widgets expose extension point APIs for customization. Developers can build customized IBM Case Manager page widgets by inheriting from the page widgets provided with IBM Case Manager and overriding the extension point APIs.

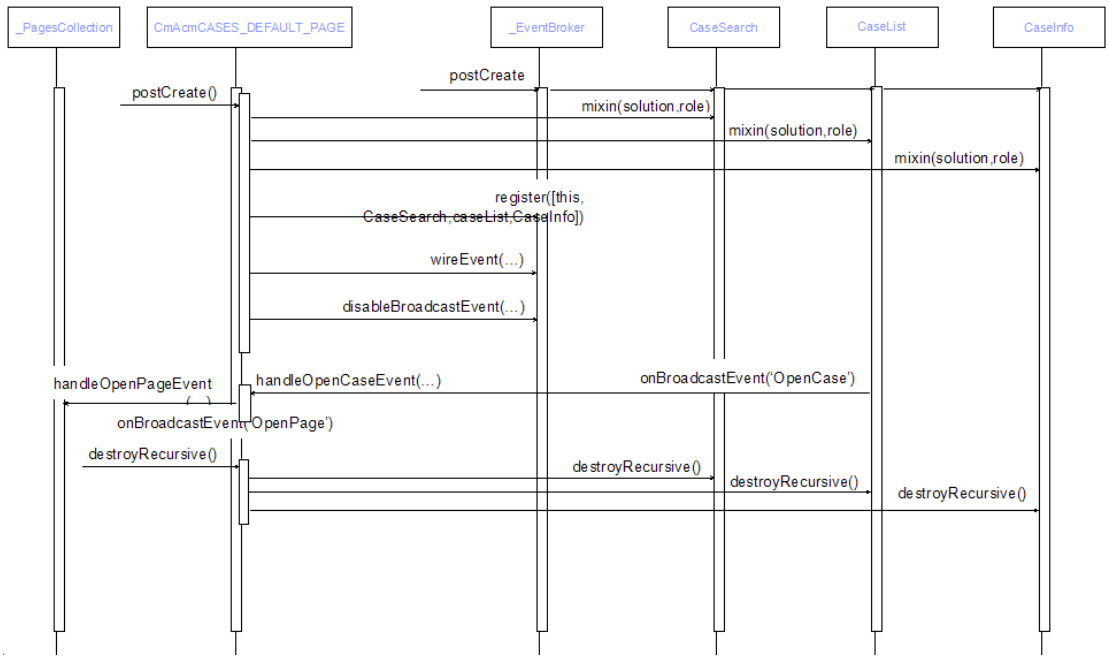
Page



Page is also a dojo dijit. It consists of three parts:

- A page dijit class inherits from icm/container/AbstractICMPage.
- An html template embeds page widgets in elements using data-dojo-type with their configuration in data-dojo-props.
- A JSON format configuration file contains the configuration of page title, event wiring, and disabled broadcast events.

The following diagram shows the sequence for page life cycle and event endpoint register.



Page is also an event endpoint. It publishes the **pages lifecycle events** to indicate some particular moments:

- The page and all page widgets are all ready to work (icm.PageOpened)
- The page is going to be closed (icm.PageClosing)
- The page is selected in a tab container (icm.PageActivated)
- The page is unselected in a tab container (icm.PageDeactivated)

In page designer, the page's layout is divided into several regions (such as top, bottom, heading, center, and trailing), and page widgets are resident in those regions. For the purpose of better managing the screen real estate to fit the user's current work, the page widget can publish events to collapse and restore a region, and the page can handle the **page layout control events**.

There is a set of broadcast events that serve the IBM Case Manager domain objects' creation and editing, such as 'icm.OpenCase', 'icm.OpenWorkItem', and 'icm.AddCase'. These events are called **command events**.

Basically, the IBM Case Manager command events are published by actions and handled by an IBM Case Manager container, which will open a proper page for processing the action according to the configuration of the current case, work item, solution, and role. With an IBM Case Manager command event broadcasting disabled, and wiring to a custom command event handler, the custom event can override the default IBM Case Manager behavior, for example, open a case in a dialog.

Container

When designing a solution, a Business analyst can associate a set of pages with a role that serves different user scenarios. Conceptually, the set of pages divide into two categories. One category is the **static page**, which is the initial set of pages the role members can start to work with, such as the case page and work page. Another category is the **dynamic page**, which can be launched from those static pages, such as the case detail page.

At runtime, the container is responsible for managing the pages. In the UI, the container has a solution and role selector to allow the user to choose the solution and role he or she wants to start work with. When the solution and role are determined, the container manages the pages in a tabbed manner. Static pages cannot be manually closed, and are there until the user switches to a different solution or role. Dynamic pages can be created for work as needed, and can be closed when the work is finished (the container will switch to the last selected page). The container also tracks the last solution and role that a user selected before exiting Case Manager Client, and opens to same solution and role the next time the user logs in.

Often, a user needs to reference a case or a work item in an email or a web page. Case Manager Client supports the ability to generate **URL addresses** in the case list and in-basket page widget. The container can digest the URL addresses to open the right detail page for the case or work item.

Action

In design time, when setting a toolbar or menu of a page widget, the action available list is shown. The actions available for setting the toolbar or menu depends on the contexts that are defined for the toolbar or menu in the page widget definition and the contexts that are defined for these actions.

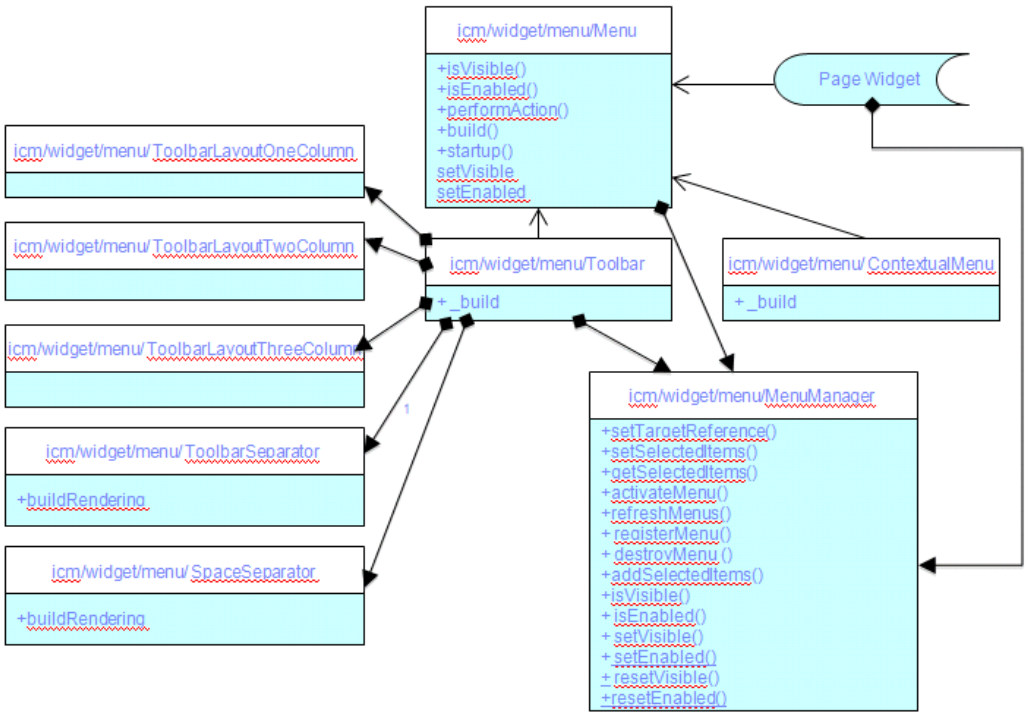
Context definition of a toolbar or menu means the toolbar and menu will provide the context resources included in the contexts definition. Context definition of an action means the toolbar and menu requires the context resources included in the context definition to be able to run.

Only the toolbar or menu can provide the context resources that the action requires. The action can be available for setting the toolbar or menu.

Like a page widget, an action can also have configurable attributes and publish events.

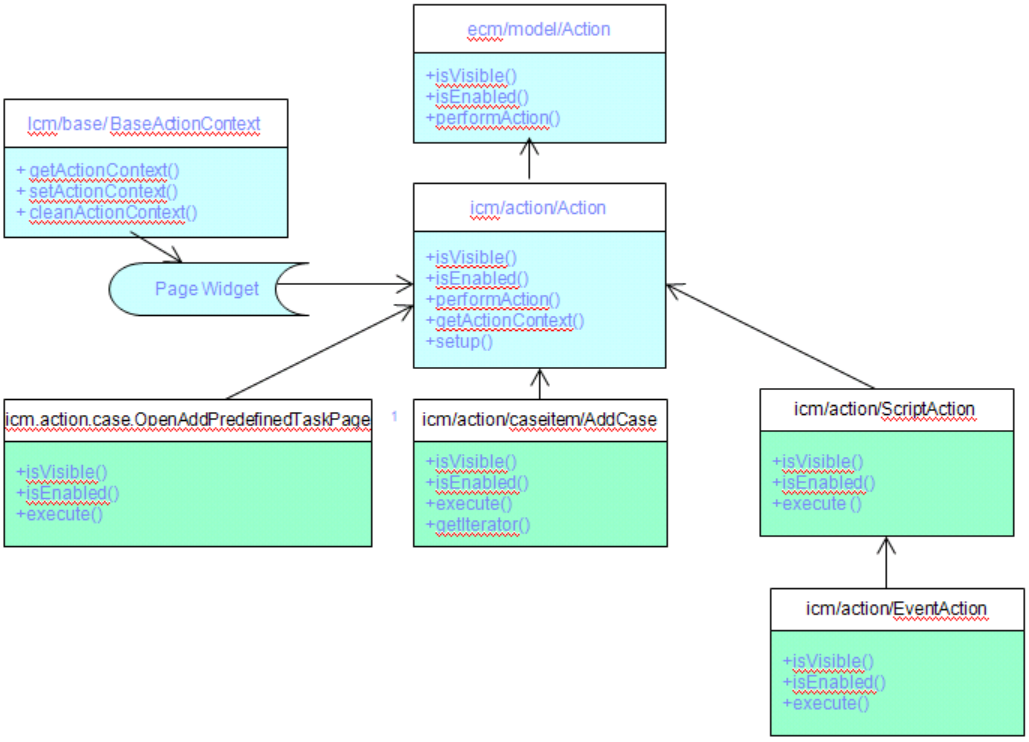
To reducing the efforts for setting the actions of a toolbar or menu, the default setting can be provided when registering a page widget. The default setting is intended to set the most usual actions that can be set in the toolbar or menu.

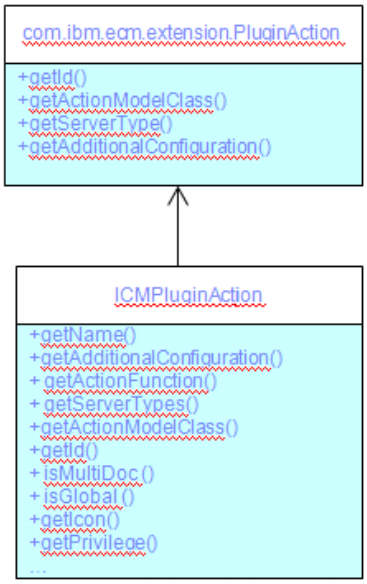
After saving the setting of actions for a toolbar or menu, the setting will be stored into page templates for a solution. The setting will be used at runtime for rendering the toolbar or menu.



When developing a customized page widget that leverages the customization capabilities of the toolbar and contextual menu, the page widget should inherit from the class icm/base/BaseActionContext, which provides a context resource pool and the functions to maintain the resource pool, such as setting, cleaning, and getting context resources from the pool, so that the page widget can maintain the context resource pool through these functions.

When developing a customized action, a JS class for the action should be developed. JS class should inherit from icm/action/Action. You could implement the performAction() method for your business operation. You also could implement the isEnabled() method and isVisible() method to control whether the buttons or menu items that your action represents are enabled or invisible.





After the action's JS class is ready, you can develop a Java class that inherits from com.ibm.ecm.extension.PluginAction. For details, refer to the developer's guide about developing a plug-in action for IBM Content Navigator.

In the Java class, you can implement the following methods for the actions, especially for Case Manager Client:

- getActionFunction() – return 'performAction'
- getActionModelClass – return the JS class name inheriting from icm/action/Action
- getAdditionalConfiguration – return the JSON object, including the properties. For explanations of JSON object properties, see Defining registry files for custom actions, properties, page widgets, and events in IBM Case Manager V5.2.

Dialog

Some actions can bring up dialogs, like the add task, add document, and add comments dialogs. Some dialogs are reused from IBM Content Navigator, such as add document. Some dialogs are built by IBM Case Manager, such as add comments.

Several dialogs are exposed as JavaScript APIs: add comments dialog, dynamic task editor dialog, add task dialog, reassign dialog, and show link dialog. You can reuse these dialogs in your custom code.

Page designer

Page layout designer is a component used in IBM Case Manager to design the UI of the page and configure the widgets included in the page. It is integrated with Case Manager Builder, and launched by Case Manager Builder when Business Analyst wants to edit a page in the solution.

The page content edit in the page layout designer is provided by Case Manager Builder, which is retrieved from the Solution Design time repository. Case Manager Builder will save the output of page layout designer into the solution repository.

A Business Analysts can give the page a title, which will show as the page's tab name in runtime. For a dynamic page, there are several options that can be the dynamic part of the page title, such as the Case ID, Case title, and Step name. In runtime, the dynamic page will be substituted by real values.

The page designer provides several **page layouts:** 3 columns, T, and I. Each layout divides the page into several regions. For example, in the I layout, the regions are top, bottom, left, center, heading, and trailing. The position of the regions regarding each other is fixed. A Business Analyst can specify the initial size of each region and mark whether the region is collapsible. A Business Analyst can select a layout for the page, and

drag and drop page widgets from a page widget palette to the regions of the page. The page widgets inside each region are stacked on each other.

For each page widget, a Business Analyst can:

- Give its height in percentage, pixels, or auto.
- Configure the edit setting for controlling its runtime behavior.
- Configure the menu and toolbar to add or remove actions.
- Disable or enable broadcast events.
- Make it a hidden page widget.

A Business Analyst can also configure event wiring between page widgets.

A page widget or an action consists of two parts, one part is the design time registry and the other part is the runtime JavaScript code. They are packaged together as an IBM Content Navigator plug-in. After registering to IBM Content Navigator, the page widgets can appear in the page widget palette of the page designer, and the action can appear in page widget's menu settings.

In the page widget registry, there are several parts of major information:

- The page widget ID, title, description, and runtime class.
- The configuration properties definitions.
- The toolbar, its context, and its default actions configuration nested in configuration properties.
- The event definitions.

Property view designer

Property view designer is a component for the Business Analyst to design a view that contains case properties in a rich layout. It is integrated with Case Manager Builder. When defining a case type, the Business Analyst can add multiple property views for it. In each view, the Business Analyst can drag and drop a different type of layout control dijit into a view (such as list, table, or tab), and drag and drop case properties into each layout control. For both layout and properties, the Business Analyst can configure attributes, such as label, size, and so on.

With a user-defined property view for a case type, the Business Analyst can assign one property view as the default view for that case type. At runtime, the property page widget will use the user-defined view as the default rather than the system-generated plain list view. Also, when designing a page, the Business Analyst can configure a property page widget in the setting pane and associate a different view for a different case type. At

runtime, the property page widget will use the configured view according to the case type of the received case.

Customizable label localization

In design time, there are always some localizable strings input by the Business Analyst, like the page title and action label in page designer and the property group name in properties designer. IBM Case Manager provides an approach to allow localization of solution-specific strings. For any property that supports localizable in its property definition, you must include isLocalized equals true.

After saving your page or property view, a resource file will be generated for those localized properties in the nls folder under the solution folder in the design object store. This resource file is in the format of a normal dojo resource bundle. In the resource file, both the key and the value are the strings input by the designer.

For each language that you want to support, download the resource file, translate the values to a different language, and then copy the file back in to a subfolder of the nls folder, where each subfolder represents a different locale.

After the solution is deployed, the client code will be able to access the localized customized strings at runtime. For more detailed steps, see Translating custom strings.

Notices

This information was developed for products and services offered in the U.S.A.
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.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, 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
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
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.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Trademarks
The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:
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.

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

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

Privacy policy considerations

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use cookies that collect each user’s user name for purposes of session management, authentication, and enhanced user usability. These cookies cannot be disabled.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, See IBM’s Privacy Policy at http://www.ibm.com/privacy and IBM’s Online Privacy Statement at http://www.ibm.com/privacy/details, the section entitled “Cookies, Web Beacons and Other Technologies” and the “IBM Software Products and Software-as-a-Service Privacy Statement” at http://www.ibm.com/software/info/product-privacy.