# iWidget Specification v1.0

**Abstract:**

A number of browser-based frameworks, each with their own component model, have been independently developed to address the need for highly interactive pages that aggregate pre-built components. This has resulted in a direct tie of components to the framework defining the component model. This loss of interoperability between components and frameworks introduces an unnecessary issue for users seeking either to direct one another to useful components or to migrate their usage of a component to a different framework. This specification defines a simple component model and explains how it can be extended such that interoperability is enhanced without the loss of more powerful extended features some frameworks choose to support.

---

## Table of Contents

---

# 1 Introduction

This specification defines a simple and extensible browser-based component model for frameworks presenting such components to a user. In this document, we call these components "`iWidgets`," although frameworks implementing this model may call them by other names. There are a number of frameworks implementing this paradigm today with significant overlap in the concepts embedded within their component models. Due to the lack of open standards, these shared concepts are all surfaced in different manners, eliminating interoperability of the components between frameworks. This specification seeks to remedy this situation by defining an open standard that surfaces these shared concepts while also providing an extensibility model for surfacing other more advanced concepts.

This specification accounts for the fact that varying technologies can be used for related server-side componentry and actively seeks to be independent of any particular such technology, as is appropriate for such Web-oriented standards. It is also noted that a number of significant script libraries have become popular. This specification does not favor any such library over another, but instead seeks to address any concerns regarding implementing the standard by leveraging these libraries.

---

# 2 Terminology

The key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in **[RFC2119]**.

## 2.1 Defining terms

The following terms are used with the supplied definitions:

- **`page`**: The larger entity being composed for presentation to a user. This may include both display and non-display items, each of which may come from a different source. The composition can be changed while it is being presented (for example new items added to the composition) such that the exact definition of when a transition to a new page occurs is a policy decision left to the implementation that is managing the composition.
- **`iWidget`**: A browser-oriented component, potentially extending a server-side component, that provides either a logical service to the page or a visualization for the user (normally related to a server-side component or a configured data source).
- **`iContext`**: Those components that provide the overall management of the page, or some portion thereof. This includes any page level controls, page layout, coordination between the various items on the page and providing the services defined in this specification to items on the page.
- **`Encapsulation wrapper`**: The portion of the overall `iContext` that encapsulates a particular `iWidget`. Whether or not this conceptual portion of the overall model results in any actual implementation depends in a large measure on the implementation choices of the particular `iContext`. Examples where no explicit encapsulation layer is needed include when each `iWidget` is loaded into its own IFrame or operating system process (e.g. XULRunner or other Desktop platforms). Other implementation choices may result in some object at this level of the conceptual model (for example merging `iWidgets` into a single page's DOM). Logically, all interactions of the encapsulated `iWidget` with the `iContext` are provided to the `iWidget` by this component. All support provided by the `iContext` for assisting the `iWidget` in dealing with its encapsulation issues are also provided by this component.
- **`Markup fragment`**: A view that the `iWidget` produces for the user. While it is possible for an `iWidget` to

simultaneously present multiple fragments to the user, this specification only deals with how the primary view gets incorporated into the overall page.

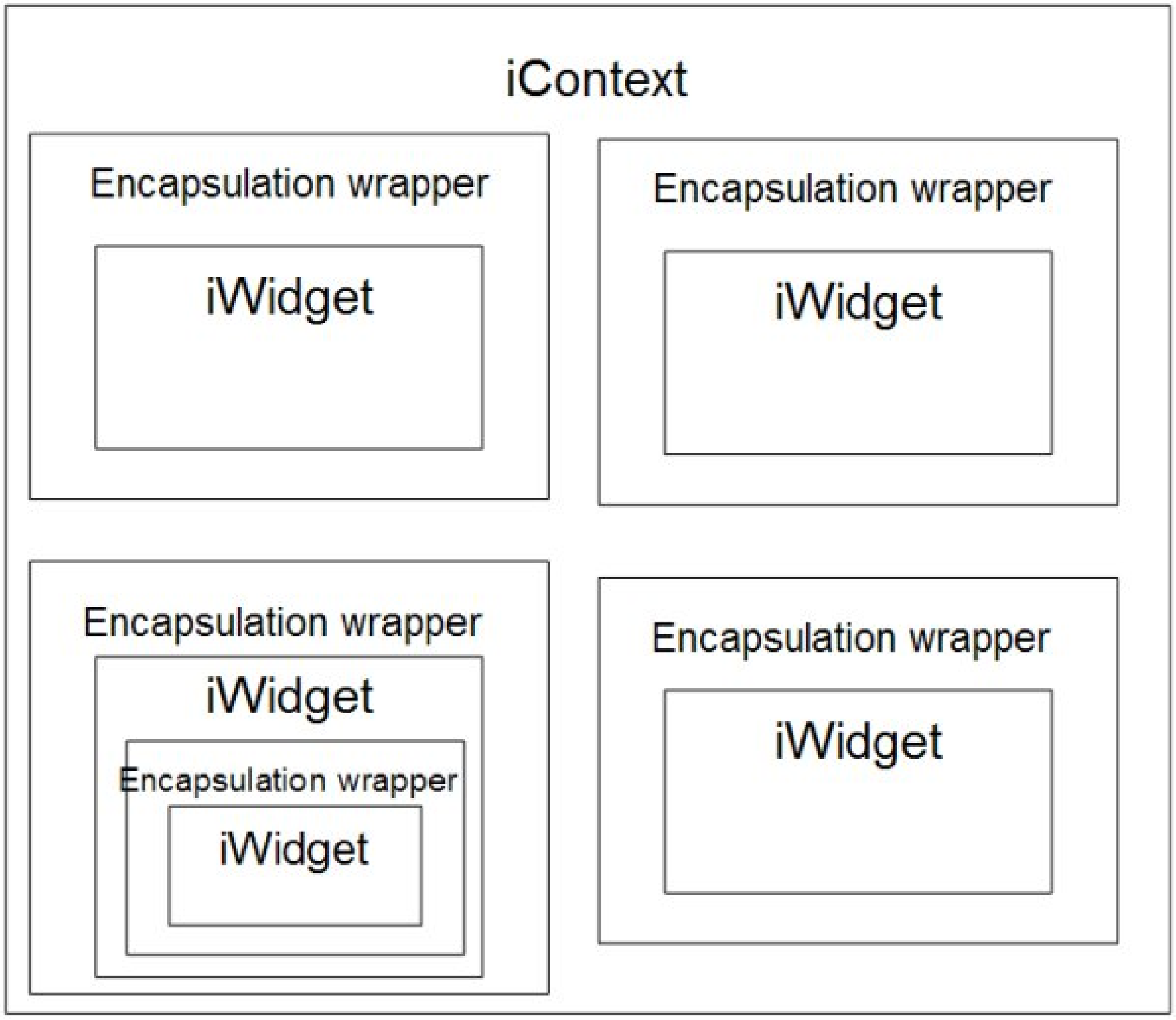
# 3 Design

The major design goals for this specification are as follows:

1. Simplicity: It should be simple to write a base-level `iWidget`, and adding more advanced features to it should be an incremental process.
2. Extensibility: There needs to be a well-defined means for `iWidgets` to determine what optional feature an `iContext` supports. There also needs to be a well-defined means for an `iContext`, which dynamically loads advanced features as they are needed, to detect what features an `iWidget` can leverage.
3. Independence: The `iContext` requirements should be straightforward to implement using available script libraries, browser extension frameworks, or application frameworks. The `iWidget` should not need to know what technology was used to implement the iContext.
4. Style: This specification will define both JavaScript (also known as EcmaScript) interfaces and at least one declarative markup that can be used to author `iWidgets`.

## 3.1 Model

The following diagram depicts how the various components defined in this specification relate to the user's visual page:



This specification seeks to enable the nesting of `iWidgets`, but only requires minor assistance of `iContext` implementations (see Section 10).

## 3.2 Concepts

This specification uses the following concepts:

- **iWidget Attribute:** The attributes of an `iWidget` typically control its look and feel as well as its data sources. An example of the first would be an attribute named "bgColor" for controlling the background color of the `iWidget`. An example of the latter would be an attribute named "feedURL" for controlling the Atom feed the `iWidget` connects to and renders. There is a special class of attributes (`iDescriptor`) described in [Section 5.2] that the `iContext` also needs to understand, for example "size".
- **Mode:** This is a per `iWidget` piece of `iContext` managed state that tells the `iWidget` what type of markup to generate for the user. Examples of the types include "`view`": normal user markup; "`edit`": markup for editing the `iWidget's` attributes; and "`help`": markup providing the user with assistance in using the `iWidget`.
- **Coordination:** This specification defines two means by which coordinated behavior can be achieved; namely:
    1. **Events:** This mechanism provides a transitory means of sharing information. It is particularly appropriate when either just a signal needs to be communicated (for example the "onLoad" event specified in [Section 6.1.3]) or the information supplied in the payload has meaning only when the event is being distributed (for example the "onItemSetChanged" event specified in [Section 6.1.3]). Information meaningful over a longer time period can also be shared using this mechanism, but it loses the querability and tolerance to fault recovery that is naturally available in the shared state mechanism.
    2. **Shared State:** This mechanism provides means (`ItemSet` definition) that an `iWidget` can use to declare state that is sharable with other components on the page. Whether or not they actually are shared will depend on the ability of other components to consume them (either as state or via events) and any policy the `iContext` applies to the sharing. An `iWidget` can become informed when other components change its shared state by registering a listener on its `ItemSets`.

## 3.3 Extensibility

This specification reserves the term "`iContext`" as the base for its provided functionality. Those defining extensions to this functionality SHOULD define the extension relative to this global term and identify the extension by naming it in the following manner:

- Use '`_`' as the first character, as this will eliminate potential conflicts with later versions of this specification
- Define a single object off `iContext` for holding all extensions defined by the same group.
- Name the extension.

As an example of how to use this methodology, suppose the JSR 286 (portlet API v2) Expert Group wishes to define extensions relative maintaining the Java portlet coding model as portions of both the portlet and portal components execute on the client. One such extension might relate to the URLGenerator concept. This extension could be named: `iContext._jsr286.urlGenerator`

# 4 Base Types

This specification uses the following types in various places.

## 4.1 ItemSet Type

This definition provides an interface to a simple abstraction of a datastore. This provides a base from which more sophisticated data stores can be built.

```
module ItemSet
{
  ItemSet            setItemValue(in String itemName, in Object value,
                                  in boolean readOnly /*optional*/);
  Object             getItemValue(in String itemName);
  ItemSet            removeItem(in String itemName);

  String[]           getAllNames();

  ItemSetDescription getItemSetDescription();

  boolean            isReadOnly(in String itemName);

  boolean            addListener(in Function listener);
  boolean            removeListener(in Function listener);

  ItemSet            clone();
}
```

Members:

- `setItemValue`: This method sets an item within the `ItemSet`, creating or replacing an existing entry as needed. Since only one value is allowed for any one name, the supplied value replaces any existing value for the supplied name. Optionally marking an item as readOnly indicates to the `iContext` that while this item may be shared with other components on the page, the access of those components SHOULD not include changing or removing the item. When successful, this method MUST return a handle to the `ItemSet` while on failure it MUST return null.
- `getItemValue`: This method returns the value for the named item from the set. On failure, this method MUST return null.
- `removeItem`: This method removes the named item from the set. When successful, this method MUST return a handle to the `ItemSet`. On failure, it MUST return null.
- `getAllNames`: This method returns an array of strings, providing the name each item currently in the set. If the set contains no items, this method MUST return null.
- `getItemSetDescription`: This method returns an `ItemSetDescription`, if one exists, for the `ItemSet`.
- `isReadOnly`: This method returns a boolean indicating whether or not the item specified by the supplied name can be modified by the user (or code acting on their behalf).
- `addListener`: This method returns a boolean indicating whether or not the request to add a listener to changes in the `ItemSet` was successful. Note that the signature for such a listener is:

```
module
{
  null  listener(in

iEvent ev);
}
```

- `removeListener`: This method returns a boolean indicating whether or not the request to remove a listener was successful.
- `clone`: This method returns a new `ItemSet` that is a duplicate of the current `ItemSet`. While this method does provide for cloning all the values within the `ItemSet`, in general this will only clone the data fields for complex Objects, both type information and any embedded logic will most likely be lost.

## 4.2 ManagedItemSet Type

This definition provides the variant on the `ItemSet` interface that is used for those `ItemSets` that the `iContext` manages.

```
module ManagedItemSet
{
  ManagedItemSet     setItemValue(in String itemName, in Object value,
                                  in boolean readOnly /*optional*/);
  Object             getItemValue(in String itemName);
  ManagedItemSet     removeItem(in String itemName);

  String[]           getAllNames();

  boolean            isReadOnly(in String itemName);

  null               save(in Function callbackFn /*optional*/);

  boolean            addListener(in Function listener);
  boolean            removeListener(in Function listener);

  ManagedItemSet     clone();
}
```

Members:

- `setItemValue`: This method sets an item within the `ManagedItemSet`, creating or replacing an existing entry as needed. Since only one value is allowed for any one name, the supplied value replaces any existing value for the supplied name. Note that while an arbitrary Object can be specified for the value, the `iContext` will only persist data fields and therefore any functions defined on the Object will be lost. It is also likely that any specific type used will also be lost across the persistence operation. Optionally marking an item as readOnly indicates to the `iContext` that while this item may be shared with other components on the page, the access of those components SHOULD not include changing or removing the item. When successful, this method MUST return a handle to the `ManagedItemSet`. On failure, it MUST return null.
- `getItemValue`: This method returns the value for the named item from the set. On failure, this method MUST return null.
- `removeItem`: This method removes the named item from the set. When successful, this method MUST return a handle to the `ManagedItemSet`. On failure, it MUST return null.
- `getAllNames`: This method returns an array of strings, providing the name each item currently in the set. If the set contains no items, this method MUST return null.
- `isReadOnly`: This method returns a boolean indicating whether or not the item specified by the supplied name can be modified by the user (or code acting on their behalf).
- `save`: This method requests the `iContext` save the current state of the `ManagedItemSet`. The `iContext` MAY also choose to save the `ManagedItemSet` at times other than when `iWidgets` request a save. This method MAY operate asynchronously. The `iContext` MUST invoke any supplied callbackFn upon completion of the save attempt. The signature for the function is:.
  ```
  function(in String managedItemSetName, in boolean success);
  ```
- `addListener`: This method returns a boolean indicating whether or not the request to add a listener to changes in

the `ItemSet` was successful. Note that the signature for such a listener is:

```
module
{
  null  listener(in


iEvent ev);
}
```

- `removeListener`: This method returns a boolean indicating whether or not the request to remove a listener was successful.
- `clone`: This method returns a new `ManagedItemSet` that is a duplicate of the current `ManagedItemSet`. While this method does provide for cloning all the values within the `ManagedItemSet`, in general this will only clone the data fields for complex Objects, both type information and any embedded logic will most likely be lost.

## 4.3 ItemSetDescription Type

This interface provides access to descriptive metadata concerning an `ItemSet`.

```
module ItemSetDescription
{
  String          name;
  ItemDescription items[];

  String          getDescription(in String locale);
}
```

Members:

- `name`: This field provides the name of the `ItemSet` being described.
- `items`: This field provides an `ItemDescription` for each item within the `ItemSet`.
- `getDescription`: This method returns a user-oriented description (markup allowed) of the `ItemSet` in the requested locale. If no locale is supplied, the default locale of the `iWidget` is used (with "en" being the default `iWidget` locale).

## 4.4 ItemDescription Type

This interface provides access to descriptive metadata concerning an item within an `ItemSet`.

```
module ItemDescription
{
  String          name;
  String          type;

  String          getDescription(in String locale);
}
```

Members:

- `name`: This field provides the name of the item being described.
- `type`: This field provides the type of the item being described. In addition to referencing JavaScript types, other standard type definitions or means of defining types (such as XML schema), `iWidget` can use the following JSON syntax to describe JSON objects: "{field1name:field1type, field2name:field2type, ...}". Such descriptive objects can be named JSON objects, loaded as part of a referenced resource and then referenced by name.
- `getDescription`: This method returns a user-oriented description (markup allowed) of the item in the requested locale. If no locale is supplied, the default locale of the `iWidget` is used (with "en" being the default `iWidget` locale).

# 5 iWidgets

This specification does not require `iWidgets` to conform to a specific interface. All `iContext`-initiated interactions with the `iWidget` are modeled either as an event or are mediated by an `ItemSet`. Of particular concern are the customization attributes of the `iWidget`. When managed by the `iContext`, they are modeled as an `ManagedItemSet`. When managed directly by the `iWidget`, they are unknown to the `iContext`.

## 5.1 iWidget attributes

We use the term "`iWidget` attributes" to refer to the customization points of the `iWidget`. These are often exposed to the user to provide reasonable default behavior (for example an `iWidget` that displays weather for the US might define an attribute for 'zipcode' and then use it as the default location for the displayed weather forecast). The persistence and lifecycle for these attributes are managed by the `iContext` and therefore they are exposed through a `ManagedItemSet`. The actual items within this set are determined entirely by the `iWidget`.

## 5.2 iDescriptor

To increase interoperability and eliminate potential conflicts with attribute names, this specification defines those attribute-like items that the `iContext` needs to understand in a special `ManagedItemSet` called `iDescriptor`. The `iContext` MAY persist `iDescriptor`, but at a minimum SHOULD retain the values for `iDescriptor` across page refreshes. The `iWidget` author SHOULD NOT expect them to last for longer than the current user's session. The following names are defined for this `ManagedItemSet`:

1. `title`: This attribute suggests a title that can be used in any decoration (for example titlebar) around the `iWidget`.
2. `name`: A short name for the `iWidget`. This is likely to be used if the `iWidget` is offered on a pallette of `iWidgets` for placing on the page.
3. `description`: A longer description (markup allowed) of the `iWidget`, suitable for displaying on a tooltip.
4. `defaultHeight`: For those `iWidgets` needing a specified size, this item provides a means to inform the `iContext` of a preferred initial height.
5. `defaultWidth`: For those `iWidgets` needing a specified size, this item provides a means to inform the `iContext` of a preferred initial width.
6. `locales`: A space-delimited list of the locales to use for any values being displayed to the user. This is an ordered list with earlier items being preferred over later items.
7. `mode`: The mode saying what markup the `iWidget` SHOULD be currently displaying.
8. `size`: The current size for the `iWidget`. This is specified using the format "width;height".
9. `author`: The name(s) of the `iWidget` author(s).
10. `email`: The e-mail(s) of the `iWidget` author(s).
11. `website`: A URI for the `iWidget` author's Web site.
12. `version`: The version number of the `iWidget` implementation in the form of n.m, where 'n' is the major version number and 'm' is the minor version number.
13. `globalAttributes`: These are attributes that the user wants to be applied across `iWidgets` (best practice is to look here before examining the iWidgetAttribute ItemSet).

# 6 iContext Type

This interface defines methods that the `iWidget` can use to interact with the `iContext`. Note that they will in general be provided by some wrapper type in order to specialize the functionality to the `iWidget` as part of proper encapsulation support.

```
module iContext
{
  ManagedItemSet getiWidgetAttributes();
  ManagedItemSet getUserProfile();
  ManagedItemSet getiDescriptor();
  ItemSet        getItemSet(in String name, in Boolean private);

  Object         iScope();

  String         processMarkup(in String markup);
  null           processiWidgets(in DOMNode node);

  Element        getElementById(in String id);
  Element[]      getElementByClass(in String className);
  Element        getRootElement();

  null           requires(in String requiredItem,
                          in String version /*optional*/,
                          in String uri,
                          in Function callbackFn /*optional*/);

  iEvents        iEvents;
  IO             io;
  xml            xml;
}
```

Members:

- `getiWidgetAttributes`: This method returns the `ManagedItemSet` that provides access to the `iWidget's` customization attributes. If there is no `ManagedItemSet` related to the `iWidget's` customization attributes, this method MUST create an empty set and return it.
- `getUserProfile`: This method returns the `ManagedItemSet` that provides access to the user's profile data. If there is no `ManagedItemSet` related to the user's profile, this method creates an empty set and returns it. If access to the user's profile is denied, this method returns null. While it is a matter left to the `iContext's` policy, it is likely that most items within the user profile will be marked as "readOnly".
- `getiDescriptor`: This method returns the `ManagedItemSet` that provides access to attributes that both the `iContext` and the `iWidget` need to understand. If there is no `ManagedItemSet` related to the `iWidget's` descriptive items, this method MUST create an empty set and return it.
- `getItemSet`: This method returns an `ItemSet` corresponding to the requested name. If it does not already exist, an `ItemSet` will be created and associated with the supplied name. If a new `ItemSet` is created, the "private"

parameter controls whether or not the `ItemSet` can be exposed to other components. If the `ItemSet` already exists, the "private" parameter is ignored. If access to the desired `ItemSet` is denied or the `ItemSet` cannot be created, null is returned.

- `iScope`: This method returns an instance of type Object that was initialized prior to the loading of the `iWidget` (either as a generic Object or an instance of the encapsulation Object referenced by declarative means (see Section 9)) . Its purpose is to support proper encapsulation of the `iWidget's` assets (variables and methods) such that multiple instances of the `iWidget` can be loaded into a single page's DOM without stepping on each other.
- `processMarkup`: This method requests the `iContext` to process the markup such that it can be inserted into the `iWidget's` markup and properly interact with the page (that is, logically extends the processing related to loading the page). Where, when, and how the markup is inserted into the page is the responsibility of the `iWidget`. On success, this method MUST return the processed markup. On failure, it MUST return null.
- `processiWidgets`: This method requests the `iContext` to process the subtree under the supplied node for the purpose of resolving and instantiating any referenced `iWidgets`.
- `getElementById`: This method provides the same semantics as the DOM method by the same name with the distinction that this method will restrict the search to the `iWidget's` markup rather than the entire page.
- `getElementByClass`: This method returns an array of Elements within the `iWidget's` markup that have the supplied value as one of those specified by the Element's "class" attribute.
- `getRootElement`: This method returns the root element of the `iWidget` such that the `iWidget` can easily do things such as searching its own markup. This also means the root element can be a convenient location to place items that the `iWidget` wishes to access later.
- `requires`: This method provides the means for an `iWidget` to declare dependency on a set of non-required items. The following names refer to optional functionality defined here:
    - io
    - xml

  For these items, no URI should be specified as the `iContext` is responsible for loading any dynamic portions of itself. In addition, sharable resources (for example "dojo.js") can be loaded just once for all `iWidgets` using the resource via this method. As there may be issues with having mulitple versions of such resources loaded at the same time, this method includes a version parameter to inform the `iContext` of any version dependency. If no value is supplied for the required version, the `iWidget` author in indicating that any version is acceptable. As `iContexts` are likely to do such dynamic loads in an asynchronous manner, the callbackFn provides a means for the `iWidget` to become aware that the required resource is available. It should be noted that the callbackFn may be invoked prior to the return from this request, especially for those resources already loaded. The callbackFn signature is of the form

  ```
  function(requiredItem, uri, resourceHandle /* when appropriate */)
  ```
- `iEvents`: This field contains an object that provides access to event services in a manner allowing the `iWidgets` on the page to interact in a loosely coupled manner while also providing control to whomever is defining the overall page/application. Types related to eventing are defined in a separate section below.
- `io`: This optional field contains an object that provides access to io services in a manner allowing the page as a whole to remain consistent, particularly in the face of server-side coordination between related application components.
- `xml`: This optional field is a placeholder for future definitions providing explicit support for XML-oriented processing. This version of the specification provides no such definitions, but they are expected in future versions.

Extensions supported by more advanced `iContexts` SHOULD follow the pattern (defined in Section 3.3) used for the iEvents and IO portions of these definitions. This allows `iWidgets` to determine the support for a "foo" extension defined by a group named "bar" using a construct of the form:

```
 var fooSupported = (iContext._bar & iContext._bar.foo);
```

## 6.1 iEvents Types

This section contains the definition of the types involved in event distribution. The model used for event distribution has the `iWidget` declare what events it can both originate and consume, and the `iContext` then wires these events together in whatever manner it chooses (examples include a backplane design that simply connects all matching events and an explicit wiring model where the page author selects the event flows that are enabled). In addition to distributing the events between `iWidgets` (or other items on the page), the `iContext` MAY choose to provide a mapping between certain events. This is frequently useful when two distinct event definitions contain the same semantic fields, but with different names at either the field or event level (or both).

The `iEvents` type has the following signature:

```
module iEvents
{
  null

publishedEvents(in iEventDescription eventDesc[]);

  null  handledEvents(in iEventDescription eventDesc[]);

  null  fireEvent(in String name, /* event name, preferrably a serialized QName */
            in String type, /* optional reference to type,
                        preferrably a serialized QName */
            in Object payload /* optional ... the event's data */);
}
```

Members:

- `publishedEvents`: This method informs the `iContext` of events that the `iWidget` might originate.
- `handledEvents`: This method informs the `iContext` of events that the `iWidget` is capable of handling.
- `fireEvent`: This method informs the `iContext` that distributing an event of the type described by the parameters is appropriate. Whether or not such an event will actually be created and distributed will depend on whether the `iContext` has any handlers for that event.

### 6.1.1 iEventDescription Type

This object contains various pieces of information describing an event.

```
module iEventDescription
{
  String   name;
  String   type;
  String[] aliases;
  String   handlingFn;
  ItemSet  handlerAttributes;

  String   getDescription(in String locale);

}
```

Members:

- `name`: The name of the event, preferably a QName serialized as "{URI}localName".
- `type`: The type of any payload. If this is set to null, no information is being provided. This is useful for both simple signals or events with opaque payloads (that is, a minor attempt at making it private). In addition to referencing JavaScript types, other standard type definitions or means of defining types (such as XML schema), `iWidget` can use the following JSON syntax to describe JSON objects: "{field1name:field1type, field2name:field2type, ...}". Such descriptive objects can be named, loaded as part of a referenced resource, and then referenced by name.
- `getDescription`: This method returns a user-oriented description (markup allowed) of the event in the requested locale. If no locale is supplied, the default locale of the `iWidget` is used (with "en" being the default `iWidget` locale). The description is likely to be displayed when a user is wiring event flow between `iWidgets`.
- `aliases`: If there are other events known to be semantically equivalent, this array provides that information as a hint to the underlying system, which might expose it to the user. This does not imply the types of the events are identical, just that the semantics are equivalent.
- `handlingFn`: Used only when describing which events an `iWidget` can handle. This provides the name of a callback function with the following signature and no return value.
  ```
  function(ievent, handlerAttributes)
  ```
- `handlerAttributes`: Used to supply a set of items to the event handler.

### 6.1.2 iEvent Type

This type carries various pieces of information when an event flows at runtime.

```
module iEvent
{
  String  name;
  String  type;
  Object  payload;
  String  source;
}
```

Members:

- `name`: The name of the event, preferably a QName.
- `type`: The type of any payload. If this is set to null, no information is being provided. This is useful for both simple signals or events with opaque payloads (that is, a minor attempt at making it private)
- `payload`: The data, if any, being provided by the source of the event.
- `source`: The `iWidget`-supplied name for the source iWidget. Note that some `iContexts` consistently null this field for security reasons.

### 6.1.3 Predefined iEvents

In order to enhance interoperability. the following events are defined by this specification. By convention, `iWidget`-provided methods with the same name as one of these events are considered the handler for that event unless the `iWidget` explicitly specifies a different event handler. The `iContext` MUST inspect `iWidgets` once they are fully loaded and support this pattern for default mapping of event handlers to the predefined events.

- `onLoad`: This event signals that the page has finished loading, including any invocations of requires(). As a simple notification, it carries no predefined payload. The `iContext` MUST support the "onLoad" event.
- `onReload`: This event signals that the page is about to reload the `iWidget`. This notification is intended to allow the `iWidget` to store any transient data such that it can be recovered by the reloaded instance. As a simple notification, it carries no predefined payload.
- `onUnload`: This event signals that the `iWidget` is about to be unloaded (commonly due to a page transition). This notification is intended to allow the `iWidget` to store any transient data such that it can be recovered by future uses of the `iWidget`. As a simple notification, it carries no predefined payload. The `iContext` MUST support the "onUnload" event.
- `onAdd`: This event signals that the `iWidget` is being added to a page. This notification is intended to allow the `iWidget` needing immediate user configuration to request a transition to edit mode. An example of such an `iWidget` would be a generic Atom feed renderer needing the URI to the feed before it can be usefully rendered. As a simple notification, it carries no predefined payload.
- `onDrag`: This event signals that the user had initiated a drag of the `iWidget`. It should be noted that only those `iContexts` supporting `iWidget` involvement in such an action will generate this event. In JSON syntax, this event's payload is:

```
{
"x": 0, //The x location of the mouse in screen coordinates.
"y": 0, //The y location of the mouse in screen coordinates.
"keys": 0, //The state of the various shift keys mouse.
}
```

  The x and y fields provide the location of the mouse relative to the containing window while the keys field provides a bit mask indicating which shift keys are currently depressed. Constants useful for extracting information from this bit mask are defined in [Section 8].

- `onDrop`: This event signals that the user has dropped the `iWidget`. It should be noted that only those `iContexts` supporting `iWidget` involvement in such an action will generate this event. In JSON syntax, this event's payload is:

```
{
"x": 0, //The x location of the mouse in screen coordinates.
"y": 0, //The y location of the mouse in screen coordinates.
"keys": 0, //The state of the various shift keys mouse.
}
```

  The x and y fields provide the location of the mouse relative to the containing window while the keys field provides a bit mask indicating which shift keys are currently depressed. Constants useful for extracting information from this bit mask are defined in [Section 8].

- `onModeChanged`: This event signals that the mode for the `iWidget` has changed. In JSON syntax, this event's payload is:

```
{
"newMode": "",
}
```

- `onSizeChanged`: This event signals that the size for the `iWidget` has changed. In JSON syntax, this event's payload is:

```
{
"newWidth": "",
"newHeight": "",
}
```

- `onNavStateChanged`: This event can be generated by either the `iWidget`, supplying a new value for its "navigational state", or by the `iContext`, signaling that some user action (for example activating a previously bookmarked page) has changed the `iWidget's` "navigational state" to a previously defined value. "Navigational state" is defined to be that portion of the `iWidget's` state that is required to rerender the current view at some later point in time (for example a weather `iWidget` might define some state that allows the user to return to a detail view of the weather forecast for a particular location). The payload of this event is the string representation of the "navigational state". How the actual state is serialized/deserialized relative to this string is the responsibility of the `iWidget` author.
- `onItemSetChanged`: This event signals that an `ItemSet` has changed and carries the type of change within the event payload. This event is only delivered to those `iWidgets` having registered a listener with the `ItemSet` the event references. In JSON syntax, this event's payload is:

  ```
  {
  "itemSetName": "",
  "changeType": "",
  "old": { },
  "new": { }
  }
  ```

  where `changeType` is one of:
    - "newItem": A new Item has been added to the set.
    - "changedValue": The value for an Item has been changed.
    - "removedItem": A new Item has been removed from the set.
- `onRefreshNeeded`: This event signals that the `iContext` considers the `iWidget` to be stale. Examples when this state could occur include when server-side coordination between components resulted in the component behind the `iWidget` to receive some coordination activity. Since the `iContext` can be aware of such states, but has no knowledge of the impacts on what the `iWidget` is presenting to the user, it generates this event to signal the situation to the `iWidget`.

# 7 IO Type

This interface defines the support the `iContext` is supplying regarding IO operations.

```
module IO
{
  XMLHttpRequest  XMLHttpRequest();

  URI             rewriteURI(in URI uri);

  /* A simple accessor */
  XMLHttpRequest  request(in requestVerb,
                          in URI uri,
                          in Function callbackFn,
                          in String message /*optional*/,
                          in [{headerName, value}] requestHeader /*optional*/);
}
```

Members:

- `XMLHttpRequest`: This object wraps the native XMLHttpRequest support to provide a consistent page in the face of asynchronous updates, especially those that may cause server-side coordination between the server-side components related to multiple `iWidgets` on the page. Note that the need to proxy URIs to domains outside the domain that sourced the page MAY also need the `iContext` to modify the specified URI prior to supplying it to the native XMLHttpRequest implementation. Note that either the `iContext` or proxy implementation MAY refuse to activate the requested URI due to security or policy reasons. All of the semantics/syntax of the native support apply with the exceptions/additions noted below:
    - None at this time.
- `rewriteURI`: This method takes a URI as a parameter and returns a URI that the browser can resolve for accessing the supplied URI. Examples of usage include resolving a relative URI against the source of the `iWidget's` definition and resolving other URIs to address any intermediate gateways, such as a proxy server.
- `request`: This convenience method creates a new XMLHttpRequest, registers any supplied callbackFn, sets it to asynchronous only if a callbackFn is provided, sets any supplied request headers and sends the supplied message as the request body if the requestVerb is "post" or "put" and executes the supplied requestVerb against the supplied URI. If the requestVerb is not one of "get", "post", "put" or "delete," then "get" is used.

# 8 Constants

```
Constants
iContext.constants.mode.VIEW = "view"
iContext.constants.mode.EDIT = "edit"
iContext.constants.mode.HELP = "help"
iContext.constants.ATTRIBUTES = "attributes"
iContext.constants.IDESCRIPTOR = "idescriptor"
iContext.constants.USERPROFILE = "userprofile"
iContext.constants.keys.SHIFT = 1
iContext.constants.keys.ALT = 2
iContext.constants.keys.CTRL = 4
iContext.constants.keys.META = 8
iContext.constants.keys.CAPSLOCK = 16
```

Members:

- `iContext.constants.mode.VIEW`: The generated markup fragment(s) should reflect normal interaction with the `iWidget.`
- `iContext.constants.mode.EDIT`: The generated markup fragment(s) should reflect `iWidget` interactions for editing `iWidget` attributes.
- `iContext.constants.mode.HELP`: The `iWidget` should generate markup fragment(s) to assist the user with interacting with the `iWidget.`
- `iContext.constants.ATTRIBUTES`: Name for the `ManagedItemSet` holding the customization attributes.
- `iContext.constants.IDESCRIPTOR`: Name for the `ManagedItemSet` holding the items describing the `iWidget`.
- `iContext.constants.USERPROFILE`: Name for the `ManagedItemSet` holding data about the user.

Note that not all `iWidgets` will support all modes, but they are defined here for interoperability of those leveraging these modes.

# 9 Declarative syntax

This section defines declarative syntaxes for an `iWidget`. At a minimum, this defines what is needed to properly initialize the `iWidget`. This purpose is met via tag definitions that form a relatively thin layer on top of the `iContext` interface methods a script-based `iWidget` would use to accomplish the equivalent initialization. Note that most of the tags are optional as they provide means to access `iContext` functionality that an `iWidget` is not required to access. The following table uses italics for optional items and encloses within curly braces ('{' and '}') descriptions of the values that are to be replaced (for example readOnly="{boolean}" gets replaced to be readOnly="true" or readOnly="false").

## 9.1 iWidget definition syntax

This section defines declarative syntaxes for defining an `iWidget` that can be consumed by any application providing a compliant `iContext` implementation (for example a Web page that has loaded such support). Since there are items within the definition that may want to be specified for multiple languages, authors are encouraged to use either UTF-8 or UTF-16 encoding for their definitions. If language is not specified in the areas below that support localization, the fallbacks are first the language specified for the entire `iWidget` (that is, the "lang" attribute on the root element) and then "en" (that is, the common default on the Internet).

| **XML style** | **XHTML style** |
|---|---|
| &lt;iw:iwidget<br>xmlns:iw="http://www.ibm.com/xmlns/prod/iWidget"<br>id="{iWidgetName}"<br>*allowInstanceContent="{boolean}"*<br>*supportedModes="{list of modes}"*<br>*iScope="{objectName}"*<br>*{[Section 5.2 items]}="{value}"*<br>*{eventName}="{handlerName}"*<br>*{attributeName}="{value}"*<br>*lang="{language}" &gt;* | &lt;!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"&gt;<br>&lt;html xmlns="http://www.w3.org/1999/xhtml"<br>xmlns:iw="http://www.ibm.com/xmlns/prod/iWidget"<br>*lang="{language}" &gt;*<br>*&lt;head&gt;*<br>*&lt;meta name="{[Section 5.2 items]}"*<br>*content="{value}" /&gt;*<br>*&lt;meta name="iScope" content="{objectName}" /&gt;*<br>*&lt;meta name="supportedModes"*<br>*content="{list of modes}" /&gt;*<br>*&lt;meta name="{eventName}"*<br>*content="{handlerName}" /&gt;*<br>*&lt;meta name="{attributeName}" content="{value}" /&gt;*<br>&lt;*link* type="{mimeType}" href="{uri}"<br>*title="{resourceName}" /&gt;*<br>&lt;*script* type="{mimeType}" src="{uri}" /&gt;<br>&lt;*style* type="{mimeType}" *title="{resourceName}" &gt;*<br>&lt;!-- in-line style definitions --&gt;<br>*&lt;/style&gt;*<br>*&lt;/head&gt;*<br>&lt;body *id="{iWidgetName}" onload="{handlerName}"*<br>*class="iw-AllowInstanceContent"*&gt; |

| | |
|---|---|
| `<!-- one per described event --> <iw:eventDescription id="{eventDescName}" payloadType="{payloadType}" description="{description}" descriptionURI="{uri}" lang="{language}" aliases="{aliases}" > <iw:alt description="{description}" descriptionURI="{uri}" lang="{language}" /> </iw:eventDescription>` | `<!-- one per described event --> <span class="iw-EventDescription" id="{eventDescName}"> <span class="iw-Type"> {payloadType} </span> <span class="iw-Description" lang="{language}" > {description} <a class="iw-DescriptionURI" href="{uri}" lang="{language}" /> </span> <span class="iw-Aliases"> {aliases} </ span> </span>` |
| `<!-- one per event --> <iw:event id="{eventName}" eventDescName="{eventDescName}" published="{boolean}" handled="{boolean}" {handlerItemName}="{attributeValue}" onEvent="{handlerName}" />` | `<!-- one per event --> <span class="iw-Event iw-Published iw-Handled" id="eventName"> <a class="iw-EventDescName" href="#{eventDescName}" /> <span class="iw-Handler"> {handlerName} </span> <!-- one per handler attribute --> <a class="iw-Item" href="# {itemName}"> {value} </a> </span>` |
| `<!-- one per ItemSet description --> <iw:itemSetDescription id="{setDescName}" description="{description}" descriptionURI="{uri}" lang="{language}"> <iw:alt description="{description}" descriptionURI="{uri}" lang="{language}" /> </iw:itemSetDescription>` `<!-- one per Item description --> <iw:itemDescription id="{itemDesc}" type="{type}" description="{description}" descriptionURI="{uri}" lang="{language}"> <iw:alt description="{description}"` | `<!-- one per ItemSet description --> <span class="iw-ItemSetDescription" id="{setDescName}" > <span class="iw-Description" lang="{language}" > {description} <a class="iw-DescriptionURI" href="{uri}" lang="{language}" /> </span> </span>` `<!-- one per Item description --> <span class="iw-ItemDescription" id="{itemDesc}" > <span class="iw-Type"> {type} </span> <span class="iw-Description" lang="{language}" > {description}` |

| | |
|---|---|
| descriptionURI="{uri}" lang="{language}" /> </iw:itemDescription> | <a class="iw-DescriptionURI" href="{uri}" lang="{language}" /> </span> </span> |
| <!-- one per instantiated ItemSet --> <iw:itemSet id="{setName}" private="{boolean}" itemSetDescRef="{setDescName}" onItemSetChanged="{handlerName}"> <!-- one per Item in the ItemSet --> <iw:item id="{itemName}" description="#{itemdesc}" readOnly="{boolean}" value="{value}" /> </iw:itemSet> | <!-- one per instantiated ItemSet --> <span class="iw-ItemSet iw-Private" id="{setName}"> <a class="iw-ItemSetDescRef" href="#{setDescName}" /> <span class="iw-onItemSetChanged"> {handlerName} </span> <!-- one per Item in the ItemSet --> <a class="iw-Item iw-ReadOnly" id="{itemName}" href="#{itemDesc}" > {value} </a> </span> |
| <!-- one per resource --> <iw:resource id="{resourceName}" src="{uri}" version="{version}" mimeType="{mimeType}" callback=""/> | <!-- one per resource --> <span class="iw-Resource" id="{resourceName}" > <a class="iw-Src" href="{uri}" /> <span class="iw-MimeType"> {mimeType} </span> <span class="iw-Version"> {version} </span> <span class="iw-Callback"> {callbackName} </span> </span> |
| <!-- one per supported mode --> <iw:content mode="{mode}" > <!-- default content for this mode --> </iw:content> | <!-- one per supported mode --> <span class="iw-Content {mode}" > <!-- default content for this mode --> </span> |
| </iw:iwidget> | </body> </html> |

## Element descriptions:

### ***Component metadata***

This specification defines a number of metadata items at the `iWidget`-level, namely:

- **iWidgetName:** This provides a means for textually representing the `iWidget` to a user. One example use would be on a tooltip when the mouse is hovering over a representation of the `iWidget` on a mashup tool's pallete of components. This item is supplied using:

  - XML-syntax: The "id" attribute on the <iw:iwidget> element.
  - XHTML-syntax: The "id" attribute on the <body> element.
- **iScope:** This provides the name of an object that is to be used instantiating an encapsulation object instance for the `iWidget`. The `iContext` will first try to resolve this from the set of referenced script files and then from the global context. If it fails to resolve, the generic Object class is used to create an encapsulation object instance. This value is specified using:
  - XML-syntax: The "iScope" attribute on the <iw:iwidget> element.
  - XHTML-syntax: As the value of the "content" attribute of a <meta> element that has a "name" attribute with a case-sensitive value of "iScope."
- **Event handlers:** Event handlers for the events defined in [Section 6.1.3] and others using the convention of having the event's name start with "on" followed by an upper case letter can be registered using:
  - XML-syntax: Attributes on the <iw:iwidget> element whose names match the event name.
  - XHTML-syntax: As the value of the "content" attribute of a <meta> element that has a "name" attribute where the value matches the event name. There are two special cases where XHTML events have a semantic overlap with the events defined in [Section 6.1.3], namely the "onload" and "onunload" events have the same semantics as the "onLoad" and "onUnload" events, respectively. For these two special cases, either means of specifying the event handler are supported (for example the event handler to be called when the `iWidget` may either be specified using the syntax defined in this specification or that defined in the XHTML specification).

  Like the "iScope" item, these object names are first resolved against the script files the `iWidget` has referenced and then the global context.
- **Descriptive items:** Values for the items defined in [Section 5.2] can be registered using:
  - XML-syntax: Attributes on the <iw:iwidget> element whose names match the defined name.
  - XHTML-syntax: As the value of the "content" attribute of a <meta> element that has a "name" attribute where the value matches the defined name.
- **Supported modes:** This metadata item provides a means for the `iWidget` to inform the `iContext` of the "modes" for which it can generate markup. If this is not supplied, the `iContext` will presume the only supported mode is "view." If it is supplied, the `iContext` might generate means in any decoration around the iWidget to assist the user in switching to one of these modes. These values are specified using a space-delimited list in the following manner:
  - XML-syntax: The "supportedModes" attribute on the <iw:iwidget> element.
  - XHTML-syntax: As the value of the "content" attribute of a <meta> element that has a "name" attribute with a case-sensitive value of "supportedModes."
- **iWidget attributes:** These provide standard means for the `iWidget` to provide a customized presentation for the user (for example user-set background color, prefered zipcode, and more). Note that there are two means of defining `iWidget` attributes; in addition to the one specified here, full definitions can be defined using the `ItemSet` technique defined further down for the set named "attributes." The short-hand technique does not allow specifying the read-only flag. The short-hand technique is specified using:
  - XML-syntax: Attributes on the <iw:iwidget> element that do not match any of the above items.
  - XHTML-syntax: As the value of the "content" attribute of a <meta> element that has a "name" attribute specifying a value not matched by any of the above items.
- **Instance content:** This concept is related to the nesting of `iWidgets` and is described in Section 10.

## *Event metadata*

This specification defines a number of event metadata items, namely;

- **eventName:** This provides the name that defines the semantics of the event and becomes the means by which it is refered to elsewhere. It is specified in the following manner:
  - XML-syntax: Carried as the value of the "id" attribute on the <iw:eventDescription> element.
  - XHTML-syntax: Carried as the value of the "id" attribute on a <span> element that has a value of "iw-EventDescription" for the "class" attribute.
- **payloadType:** This description of the payload type is carried in the following manner:

    - XML-syntax: Carried as the value of the "payloadType" attribute on the <iw:eventDescription> element.
    - XHTML-syntax: Carried as the value of the <span> element that has a value of "iw-Type" for the "class" attribute.
- **description:** This item provides a localized, user-oriented description of the semantics of the event. Intended uses include display on a tooltip while the user's pointer is hovering of over some display of the event relative to the `iWidget`. It is specified in the following manner:
    - XML-syntax: Carried as the value of the "description" attribute on the <iw:eventDescription> element with the language of the description specified using the "lang" attribute. Descriptions for additional languages can be supplied using child <iw:alt> elements.
    - XHTML-syntax: Carried as the value of the <span> element that has a value of "iw-Description" for the "class" attribute with the language of the description specified using the "lang" attribute. Descriptions for additional languages can be supplied by repeating this element for each supplied language.
- **aliases:** This item defines other events known to be *semantically* equivalent to this event. Whether or not there also exists a syntactic equivalence can only be determined by examining the respective "payloadTypes." It is specified in the following manner:
    - XML-syntax: Carried as a space delimited list within the value of the "alias" attribute on the <iw: eventDescription> element.
    - XHTML-syntax: Carried as the value of the <span> element that has a value of "iw-Aliases" for the "class" attribute.

## *Event specification*

Events can be specified relative to both those an `iWidget` publishes and those it can process, including the case where the same event is both published and processed. Events are declared using the <iw:event> element in the XML syntax with separate attributes for "published" and "handled" declaring the direction the event can flow. In the XHTML systex, an <a> element with a class attribute whose value may contain either or both "iw-PublishedEvent" and "iw-HandledEvent" serves the identical function.

- **Event name:** This item refers to both the name the event will specify at runtime and the name provided in any accompanying EventDescription. If no EventDescription is provided, it MUST be assumed that the event is opaque and any issues related to processing it become the responsibility of the receving component. It is specified in the following manner:
    - XML-syntax: Carried as the value of the "name" attribute.
    - XHTML-syntax: Carried as the value of the "href" attribute, prepended with '#' as this refers to a name defined by an EventDescription.
- **Event handler:** This item specifies the name of the method for processing this event. It is not valid on declarations of events that are only published. It is specified in the following manner:
    - XML-syntax: The value of the "onEvent" attribute.
    - XHTML-syntax: Carried as the value of the <span> element that has a value of "iw-Handler" for the "class" attribute.

## *ItemSet metadata*

This specification carries a number of ItemSet metadata items, namely;

- **Description name (setDescName):** This name provides the means for a particular ItemSet instance to refer back to its description. The separate name is used as there are use cases for multiple ItemSet instances based off the same description. It is specified in the following manner:
    - XML-syntax: Carried as the value of the "id" attribute.
    - XHTML-syntax: Carried as the value of the "id" attribute.
- **Description:** This item provides a localized, user-oriented description of the semantics of the ItemSet. Intended uses include display on a tooltip while the user's pointer is hovering of over some display of the ItemSet relative to the `iWidget`. It is specified in the following manner:

  - XML-syntax: Carried as the value of the "description" attribute on the <iw:itemSetDescription> element or as the content returned from a URI referenced by the "descriptionURI" attribute with the language indicated by the "lang" attribute in both cases. Descriptions for additional languages can be supplied using child <iw:alt> elements.
  - XHTML-syntax: Carried as the value of the <span> element that has a value of "iw-Description" for the "class" attribute or as the content returned from a URI referenced by the "href" attribute of an <a> element where the class attribute has the value "iw-DescriptionURI" with the language of the description specified using the "lang" attribute in both cases. Descriptions for additional languages can be supplied by repeating this element for each supplied language.
- **Item description:** This provides a localized, user-oriented description of the semantics of the Item. Intended uses include display on a tooltip while the user's pointer is hovering of over some display of the Item relative to the `iWidget`. It is specified in the following manner:
  - XML-syntax: Carried as the value of the "description" attribute on the <iw:itemDescription> element or as the content returned from a URI referenced by the "descriptionURI" attribute with the language indicated by the "lang" attribute in both cases. Descriptions for additional languages can be supplied using child <iw:alt> elements.
  - XHTML-syntax: Carried as the value of the <span> element that has a value of "iw-Description" for the "class" attribute or as the content returned from a URI referenced by the "href" attribute of an <a> element where the class attribute has the value "iw-DescriptionURI" with the language of the description specified using the "lang" attribute in both cases. Descriptions for additional languages can be supplied by repeating this element for each supplied language. A name for this description can be supplied using the "title" attribute of the <span> element with a value of "iw-Description" for the "class" attribute.
- **Item type:** This provides the syntactic information required to access the item. It is specified in the following manner:
  - XML-syntax: Carried in the "type" attribute of the <iw:itemDescription> element.
  - XHTML-syntax: Carried as the value of the <span> element that has a value of "iw-Type" for the "class" attribute.

## *ItemSet specification*

Multiple ItemSets can be instantiated from a single ItemSetDescription. The following provides the means to optionally refer back to the ItemSetDescrition while also providing values for the Items contained within the set.

- **Set name:** This provides the unique identifier for this ItemSet. It is specified in the following manner:
  - XML-syntax: Carried as the value of the "id" attribute.
  - XHTML-syntax: Carried as the value of the "id" attribute.
- **Description reference:** This item provides a means to refer back to an ItemSetDescription for the ItemSet being defined. It is specified in the following manner:
  - XML-syntax: Value of the "description" attribute.
  - XHTML-syntax: Carried in the "href" attribute of an <a> element where the class attribute has the value "iw-ItemSetDescRef," prepended with a '#' as this refers to a name defined within the same file.

  If the description reference is missing, the `iWidget` author is choosing not to supply the metadata a page author might use to connect the `iWidget` to other items on the page.
- **Private:** This boolean controls whether or not the ItemSet is sharable with other components. It is specified in the following manner:
  - XML-syntax: Carried as the value of the "private" attribute.
  - XHTML-syntax: Specified by including the "iw-Private" CSS class on the element defining the instance of an ItemSet.
- **Item(s):** This provides the means to declare individual items as part of the set. It is specified in the following manner:
  - XML-syntax: Each item is separately specified using a <iw:item> element.
    element within the <iw:ItemSet>element. The item is named using the "id" attribute, a value supplied using the "value" attribute, the description referenced using the "description" attribute, and the readOnly indicator set using the "readOnly" attribute.

- XHTML-syntax: Each item is separately specified using an <a> element where the class attribute has a value of "iw-Item," the "href" attribute references the description name for the item-level metadata, the "id" attribute names the item, and the element value provides the name to use for the item. The readOnly flags are set using the iw-ReadOnly CSS class.

Note that if the element specifying a value for an Item does not refer back to an itemName defined in the ItemSetDescription, then no metadata about the Item is being provided.

- **Change event:** When an ItemSet undergoes changes (new items, changed values, etc), it will source an onItemSetChanged event to all registered listeners other than the component which originated the change. Listeners can be registered either from the declarative markup (indicate a listener using the onItemSetChanged attribute/class) or by invoking the addListener method on the ItemSet.

## ***Resource references***

Common markup resources includes scripts, CSS definitions, images, and more. While these can be referenced in the standard XHTML manner when using theat syntax, there is also a provision for referencing resources not predefined by the XHTML specification. This may also be useful for resource used in a dynamic manner.

- **Resource name:** This provides a generic means to refer to a resource regardless of the URI from which it is loaded. The preferred form of this name is a serialized QName where the URI portion of the name provides the originating source for the resource, and the local part provides the short name of the resource. For example, the preferred resourceName for dojo.js is "{http://www.dojotoolkit.org}/dojo.js." This form of reference allows sharable resources to be loaded once by the `iContext` and used by any `iWidget` referencing them. It is specified in the following manner:
    - XML-syntax: The resource's name is specified using the "id" attribute on a <iw:resource> element.
    - XHTML-syntax: The resource's name is specified using the "id" attribute on a <span> element with a class attribute whose value is "iw-Resource."
- **Source (URI):** This defines the URI from which the resource can be fetched. Note that for shared resources, it might be fetched from a different URI (for example one provided by some other component). It is specified in the following manner:
    - XML-syntax: The URI is specified using the "src" attribute on a <iw:resource> element.
    - XHTML-syntax: The URI is specified using the "href" attribute on an <a > element with a class attribute whose value is "iw-Src."
- **Mime type:** This item tells the `iContext` how to request the resource. It is specified in the following manner:
    - XML-syntax: The MIME type is specified using the "mimeType" attribute on a <iw:resource> element.
    - XHTML-syntax: The MIME type is specified as the value of a <span> element with a class attribute whose value is "iw-mimeType."
- **Version:** This potentially allows the `iContext` to distinguish requirements for different versions of a shared resource. Loading distinct versions of some resources might not be possible due to issues it would cause in the underlying environment (for example name clashes on loading two versions of most script libraries in a browser environment). It is specified in the following manner:
    - XML-syntax: The version is specified using the "version" attribute on a <iw:resource> element.
    - XHTML-syntax: The version is specified as the value of a <span> element with a class attribute whose value is "iw-version."
- **Callback:** This item allows the `iWidget` author to specify a method that the `iContext` will invoke once the resource is loaded. It is specified in the following manner:
    - XML-syntax: The callback is specified using the "callback" attribute on a <iw:resource> element.
    - XHTML-syntax: The callback is specified as the value of a <span> element with a class attribute whose value is "iw-callback."

## ***Markup***

The `iWidget` definition can include markup for the modes that it declared to support. Regardless of whether a mode change happens due to an `iContext`-provided means or by `iWidget` actions, these different markup sections are hidden or shown depending on whether they apply to the current mode or not. The different markup sections are specified using:

- XML-syntax: Each section is contained by an <iw:content> element with a "mode" attribute whose value provides the mode where the markup section applies.
- XHTML-syntax: Each section is contained by an <span> element with a class attribute where one of its values is "iw-Content," and another provides the mode where the markup section applies.

It should be noted that the concept of modes can be used for values not defined in this specification. For example, an `iWidget` that supports ordering items may define a "viewCart" mode for the markup to show when users want to see what they have placed into their shopping cart already. The `iContext` might not provide a means for the user to switch into this mode, but will do the hide/show logic when the `iWidget` sources a change in the mode (that is, changes the value of the mode item within the `iDescriptor` set (see [Section 5.2])).

Since the `iWidget's` markup is likely to reference resources (for example an <img> element referencing an image relative to the deployment of the `iWidget`), and these relative URIs are not likely to resolve correctly relative to the base URI of the page (that is, what the browser will try), a CSS class is defined (**iw-RewriteURI**) for having the `iContext` resolve these URIs into absolute URIs whenever it is processing the markup (that is, either on page load or an invocation of processMarkup()).

Since the `iWidget's` markup is likely to reference relative resources (for example an <img> element referencing an image relative to the deployment of the `iWidget`), and these relative URIs are not likely to resolve correctly relative to the base URI of the page (for example what the browser will try), a JavaScript URL leveraging the rewriteURI() method off the "io" field of `iContext` (e.g. src="javascript:iContext.io.rewriteURI(relativeURI)").

- XML-syntax: Each section is contained by an <iw:content> element with a "mode" attribute whose value provides the mode where the markup section applies.
- XHTML-syntax: Each section is contained by an <span> element with a class attribute where one of its values is "iw-Content" and another provides the mode where the markup section applies.

## 9.2 Including iWidgets

This section defines declarative syntaxes for placing an instance of an `iWidget` into a broader set of markup (for example a Web page).

| **XML style** | **Microformat style** |
|---|---|
| <iw:iwidget id="{instanceID}" xmlns:iw="http://www.ibm.com/xmlns/prod/iWidget" *widget="{uri}"* > | <span class="iw-iWidget" id="{instanceID}"> <*a class="iw-Definition"* href="{uri}" /> |
| <!-- Override values from iWidget definition --> <*iw:itemSet* name="attributes\|{setName}" > <!-- one per item value supplied --> <iw:item name="{itemName}" value="{value}" /> *</iw:itemSet>* | <!-- Override values from iWidget definition --> <*span class="iw-ItemSet"* title="attributes\|{setName}"> <!-- one per item value supplied --> <a class="iw-Item" href="#{itemName}" > {value} </a> *</span>* |
| <!-- one per event to receive --> <*iw:receivedEvent* eventName="{targetEventName}" sourceWidget="{sourceWidgetID}" *sourceEvent="{sourceEventName}"*> <!-- one per overridden handler attribute --> <*iw:item* name="{itemName}" value="{value}" /> *</iw:receivedEvent>* | <!-- one per event to receive --> <*span class="iw-ReceivedEvent"* > <a class="iw-SourceEvent" href="#{sourceWidgetID}"> {sourceEventName} </a> <span class="iw-TargetEvent" > {targetEventName} </span> <!-- one per overridden handler attribute --> <*a class="iw-Item"* href="#{itemName}" > {value} </a> *</span>* |
| **<!-- IF ALLOWED BY THE iWidget DEFINITION! -->** <*iw:content* > <!-- default content for the view mode --> *</iw:content>* | **<!-- IF ALLOWED BY THE iWidget DEFINITION! -->** <*span class="iw-Content"* > <!-- default content for the view mode --> *</span>* |
| </iw:iwidget> | </span> |

### ***Referencing the iWidget***

When placing an `iWidget` on a page, both a reference to the `iWidget's` definition and a unique ID for the instance are required.

- **iWidget definition:** A URI where the definition of the `iWidget` can be fetched is required so that the `iContext` can have access to the metadata defining the `iWidget`. It is specified in the following manner:
    - XML-syntax: A URI to the definition is provided as the value of the "widget" attribute on the <iw:iwidget> element.

  - XHTML-syntax: A URI to the definition is provided as the value of the "href" attribute of an <a> element that has the value "iw-Definition" for the class attribute.
- **Unique identifier:** The `iContext` implementation needs a unique way of referring to each `iWidget` it is managing. The responsibility of ensuring the uniqueness of this identifier is left to the page author, though it is expected that many of the tools that assist in defining pages containing `iWidgets` will take care of this responsibility. It is specified in the following manner:
  - XML-syntax: The unique ID is specified as the value of the id attribute on the <iw:iwidget> element.
  - XHTML-syntax: The unique ID is specified as the value of the "id" attribute of an element that has the value "iw-iWidget" for the class attribute (normally a <span>).
- **Instance-level content:** There are use cases for an `iWidget` to provide a set of functionality that is applied to content supplied when an instance of it is placed on a page. In those cases, the `iWidget` definition specifies that instance content is allowed, and the reference that places an instance on a page specifies content in the same manner as the `iWidget` definition would have.

## ***Specifying ItemSet data***

An instance of an `iWidget` is not allowed to specify new ItemSets, but is allowed to supply and override values for the items in the `iWidget` definition.

- **Set name:** This provides the name of the ItemSet where the instance wants to set values. It is specified in the following manner:
  - XML-syntax: Carried as the value of the "id" attribute.
  - XHTML-syntax: Carried as the value of the "id" attribute.
- **Values for items:** These are provided in the following manner:
  - XML-syntax: Each item is separately specified using a <iw:item> element.
  - XHTML-syntax: Each item is separately using an <a> element whose "class" attribute has a value of "iw-Item."

## ***Specifying events to be processed***

When placing instances of `iWidgets` on a page, it is often desirable to specify the coordination behavior as well. Since this involves connecting instances on the page, this specification defines a syntax for specifying which events an `iWidget` instance should receive using the following items.

- **Received event:** This defines an event that should be distributed to this `iWidget` instance. One of these is defined for each event the `iWidget` instance should process and is specified in the following manner:
  - XML-syntax: A <iw:receivedEvent> element per event to process. Note that multiple such elements for a single event name can be specified for processing events sourced by different `iWidget` instances on the page.
  - XHTML-syntax: A <span> with a "class" attribute having the value of "iw-ReceivedEvent" for each event to be distributed to this `iWidget` instance. Note that multiple such elements for a single event name can be specified for processing events sourced by different `iWidget` instances on the page.
- **Event name:** This specifies the name by which this `iWidget` instance knows the event.
  - XML-syntax: This is carried as the value of the "eventName" attribute on the <iw:receivedEvent> element.
  - XHTML-syntax: Carried as the value on a <span> element with a "class" attribute having the value "iw-TargetEvent."
- **Event Source:** This specifies both the identifier of the `iWidget` instance that publishes the event whose routing is being defined and the name under which it is published.
  - XML-syntax: The identifier of the event source is carried as the value of the "sourceWidget" attribute on the <iw:receivedEvent> element. The name under which the event is being published is carried as the value of the "sourceEvent" attribute on the <iw:receivedEvent> element.
  - XHTML-syntax: The identifier of the event source is carried as the target of the "href" attribute on an <a>

element with a "class" attribute having the value "iw-SourceEvent." The name under which the event is being published is carried as the value of this same element.

- **Handler attributes:** This provides a means to pass a set of items into the method processing the event. They are specified in the same manner as the "Values for items" described in the previous section.

# 10 Nesting iWidgets

There are three significant use cases related to the nesting of `iWidgets` and two flags for use in providing information to the `iContext` about required support:

1. **<u>Simple container:</u>** This type of `iWidget` is not interested in being involved in the control flow relative to the `iWidgets` it contains. The kinds of functionality provided include things such as laying out the contained `iWidgets` according some predefined rules. Due to the nature of these containers, the content cannot reasonably be defined until an instance is placed on a page. In order to have the `iContext` allow this content to be specified within the instance placed on the page, the "iw-AllowInstanceContent" CSS class or "allowInstanceContent" attribute have to be set on the root element of the `iWidget` definition. Note that this use case has the `iContext` managing both the container `iWidget` and any nested `iWidgets`.
2. **<u>Defined container:</u>** This type of `iWidget` is frequently derived by a user deciding to save a set of `iWidgets` and any wiring between them. As a result, this type of container differs from the Simple Container in that the content is defined in the `iWidget` definition rather than in the instance placed on a page. As a result, neither of the flags related to nesting `iWidgets` is set. Note that this use case has the `iContext` managing both the container `iWidget` and any nested `iWidgets`.
3. **<u>Composites:</u>** This type of `iWidget` is the logical component for the content it contains, and the nature of the contained `iWidgets` is not directly visible outside the composite. While it is expected that composites will normally specify their content within their definition, the composite is also allowed to set the flag allowing instance content. Note that this use case has the `iContext` managing the composite `iWidget` and the composite managing any nested `iWidgets`. While this version of the specification provides for no assistance from the `iContext` relative to managing the nested `iWidgets`, it is expected future versions will define such assistance.

Some of the differences resulting from the above use cases include:

- When a Simple Container is dragged off a pallete and placed on a page, it usually arrives empty, and other `iWidgets` have to be dragged off the pallete to populate its content. When `iWidgets` are placed within the Simple Container, they are exposed to the overall management provided by the `iContext` in the same manner as if they were placed outside the context of a container.
- When a Defined Container or a Composite is dragged off a pallete and placed on a page, the full set of referenced `iWidgets` are also placed on the page. For the Defined Container, these `iWidgets` are separately managed, just as they are for the Simple Container while the nested `iWidgets` of a Composite only have their attributes and internal wiring exposed if the Composite chooses to do that exposure.

The semantics of the flag from the above discussion:

1. **<u>allowInstanceContent (CSS class = iw-AllowInstanceContent):</u>** This flag tells the `iContext` to replace view-mode content from the `iWidget` definition with the equivalent content from the instance on the page. This flag may only be set in the definition of an `iWidget`.

---

# Appendix A: CSS classes

The following set of CSS classes (with attribute equivalents for the XML style of declarative markup) are defined in this specification:

**Reused "utility" classes**

- **iw-Description:** The element with this CSS class provides a user-oriented description. What is being described is dependent on the subtree containing this element.
- **iw-DescriptionURI:** The element with this CSS class provides a URI where a description can be retrieved.
- **iw-Type:** The element with this CSS class defines a type relative to whatever is being described in the subtree containing this element.
- **iw-Aliases:** The element with this CSS class provides a space-delimited list of things known to be semantically equivalent to that which the containing subtree is containing.

**Event classes**

- **iwEventDescription:** The element with this CSS class is the container for the description of a single event. This element has to declare the event's name using the "id" attribute, but the rest of the descriptive information is provided by child elements.
- **iw-PublishedEvent:** The element with this CSS class declares that the event being referenced is published by the `iWidget` to inform other components of activity within the `iWidget`.
- **iw-HandledEvent:** The element with this CSS class declares that the event being referenced can be processed by the `iWidget`.
- **iw-Event:** The element with this CSS class provides the reference to an EventDescription for the event being described.

**ItemSet classes**

- **iw-ItemSetDescription:** The element with this CSS class contains a subtree that provides a description for a set of items.
- **iw-Private:** Including this CSS class when defining an `ItemSet` indicates the `ItemSet` is not to be shared with other components.
- **iw-Item:** The element with this CSS class provides a value for the item name it references.
- **iw-onItemSetChanged:** The element with this CSS class provides the handler (same semantics as iw-Handler) for the onItemSetChanged event related to the referenced ItemSet.

**Miscellaneous classes**

- **iw-RewriteURI:** The element with this CSS class has an attribute whose value is a (for example the "src" attribute on an <img> tag or the "href" attribute on an <a> tag) that needs to be resolved to a URI that the browser can load.

**Instance classes**

- **iw-iWidget:** The element with this CSS class places an `iWidget` instance on the page.
- **iw-Definition:** The element with this CSS class provides a URI to the `iWidget` definition.
- **iw-Handler:** The element with this CSS class declares the name of the method that should be called to process the event.
- **iw-ItemDescription:** The element with this CSS class contains a subtree that provides a description for a single item.
- **iw-ReadOnly:** Including this CSS class when defining an ItemDescription indicates the Item can be shared with other components, but they are not allowed to modify the item's value.
- **iw-ItemSet:** The element with this CSS class contains a subtree that defines a single ItemSet.
- **iw-ItemSetDescRef:** The element with this CSS class provides a reference to the relevant ItemSetDescription.
- **iw-Resource:** The element with this CSS class contains a subtree that defines a resource the `iWidget`.
- **iw-Src:** The element with this CSS class provides the URI for retrieving the thing described in the containing subtree.
- **iw-mimeType:** The element with this CSS class provides the MIME type of the thing described in the containing subtree.
- **iw-version:** The element with this CSS class provides the version of the thing described in the containing subtree.
- **iw-callback:** The element with this CSS class provides the name of a method to call once the resource is loaded.
- **iw-Content:** The element with this CSS class contains the content relevant to the referenced mode.
- **iw-AllowInstanceContent:** An `iWidget` definition specifying this CSS class allows an instance placed on a page to define a different content than what the definition has defined.
- **iw-ReceivedEvent:** The element with this CSS class contains a subtree defining an event that should be

distributed to the containing `iWidget` instance.

- **iw-SourceEvent:** The element with this CSS class provides the ID of the `iWidget` sourcing the event as well as the name of the event when that name is different from the event name being handled by the containing `iWidget`.
- **iw-TargetEvent:** The element with this CSS class provides the name of the event to be distributed to the containing `iWidget`.

# Appendix B: Conformance statements

| # | Section | Conformance statement |
|---|---|---|
| CS002 | 4.1 | Optionally marking an item as readOnly indicates to the `iContext` that while this item may be shared with other components on the page, the access of those components SHOULD not include changing or removing the item. |
| CS003 | 4.1 | When successful, this method *(setItemValue)* MUST return a handle to the `ItemSet`. On failure, it MUST return null. |
| CS004 | 4.1 | When successful, this method *(removeValue)* MUST return a handle to the `ItemSet`. On failure, it MUST return null. |
| CS005 | 4.1 | On failure, this method *(getItemValue)* MUST return null. |
| CS006 | 4.1 | If the set contains no items, this method *(getAllNames)* MUST return null. |
| CS007 | 4.2 | Optionally marking an item as readOnly indicates to the `iContext` that while this item may be shared with other components on the page, the access of those components SHOULD not include changing or removing the item. |
| CS008 | 4.2 | When successful, this method *(setItemValue)* MUST return a handle to the `ManagedItemSet`. On failure, it MUST return null. |
| CS009 | 4.2 | When successful, this method *(removeValue)* MUST return a handle to the `ManagedItemSet`. On failure, it MUST return null. |
| CS010 | 4.2 | On failure, this method *(getItemValue)* MUST return null. |
| CS011 | 4.2 | If the set contains no items, this method *(getAllNames)* MUST return null. |
| CS012 | 4.2 | This method *(save)* MAY operate asynchronously. |
| CS013 | 4.2 | The `iContext` MUST invoke any supplied callbackFn upon completion of the save attempt. |
| CS014 | 5.2 | The `iContext` MAY persist `iDescriptor`, but at a minimum SHOULD retain the values for `iDescriptor` across page refreshes. The `iWidget` author SHOULD NOT expect them to last for longer than the current user's session. |
| CS015 | 5.2 | The mode saying what markup the `iWidget` SHOULD be currently displaying. |
| CS016 | 6 | If there is no `ManagedItemSet` related to the `iWidget's` customization attributes, this method *(getiWidgetAttributes)* MUST create an empty set and return it. |
| CS017 | 6 | If there is no `ManagedItemSet` related to the `iWidget's` descriptive items this method *(getiDescriptors)* MUST create an empty set and return it. |
| CS018 | 6 | On success this method *(processMarkup)* MUST return the processed markup while on failure it MUST return null. |
| CS019 | 6 | Extensions supported by more advanced `iContexts` SHOULD follow the pattern (defined in Section 3.3) used for the iEvents and IO portions of these definitions. |
| CS020 | 6.1 | In addition to distributing the events between `iWidgets` (or other items on the page), the `iContext` MAY choose to provide a mapping between certain events. |
| CS021 | 6.1.3 | The `iContext` MUST inspect `iWidgets` once they are fully loaded and support this pattern for default mapping of event handlers to the predefined events. |
| CS022 | 6.1.3 | The `iContext` MUST support the "onLoad" event. |
| CS023 | 6.1.3 | The `iContext` MUST support the "onUnload" event. |