Acrobat and PDFL SDK: Tracker

This PDF is programmatically generated: Review copy only

Contents

CHAPTER

ONE

ACROBAT AND PDFL SDK: TRACKER APIS

Acrobat DC Tracker is an XML-based tool, based on RSS 2.0, for presenting and describing lists of items. RSS 2.0 is an XML-based format used for describing lists of items. It also allows for extensions at any level by using XML namespaces. You can add content to Tracker from a server using RSS.

1.1 Benefits of RSS

Information is often published using RSS because it makes it easy to alert the content consumer when changes take place. RSS can be used to improve a Web site that consumers are constantly checking or a service that sends notifications through email. The following are examples of what can be published using RSS:

- Personal email (using authentication) or the contents of an email list
- The result of a search
- A set of documents that have been published for review
- The state of an approval or form workflow
- A blog

RSS allows the content provider to add custom extensions, such as metadata beyond that defined in the RSS standard. These custom extensions can be used by customizing Tracker. The extensions customize only the appearance of the content in Tracker. The content can still be used in other desktop RSS aggregators or in a server application such as a Web portal.

Note: Tracker is designed to be used in an occasionally connected environment. The RSS data, icons, and user interface driver objects are stored persistently on the local machine. However, external references, such as images used in an RSS feed, are not stored. Keep this in mind when generating content for Tracker if working offline is required.

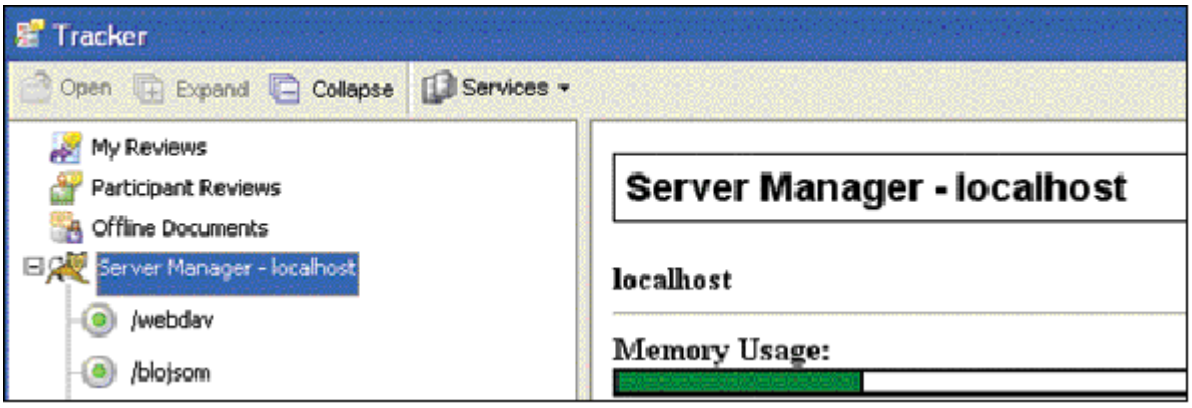
For more information on RSS, see its specification at http://blogs.law.harvard.edu/tech/rss .

1.2 Customizing the interface

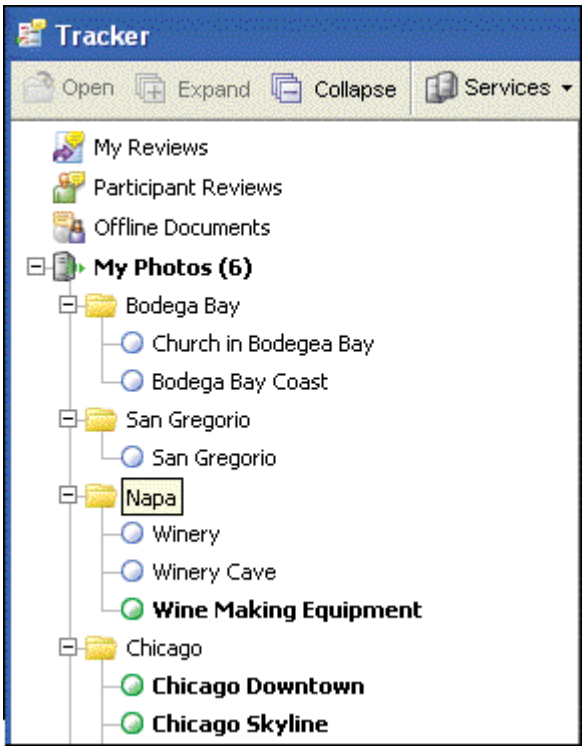
The Tracker user interface is controlled through URL commands, XML extensions used in RSS 2.0 documents, and JavaScript extensions. The Tracker URL commands manage feed subscriptions and displaying the Tracker user interface. The user interface is managed by the XML extensions, which work together with a JavaScript-based user interface driver to customize its appearance and behavior.

You can customize the Tracker user interface by adding controls such as toolbar buttons and menus, and grouping feed items as shown in the following graphics.

Using RSS extensions to customize how icons are displayed



Dynamically changing grouping categories



For information on customizing the functionality of Tracker, see Tracker API, and Customization Examples.

CHAPTER

TWO

TRACKER API

This chapter presents information on the Tracker URL API, XML feed extensions, and a JavaScript-based user interface driver for customizing the Tracker user interface.

2.1 Tracker URL API

Tracker responds to the execution of URL commands for adding, updating, selecting, and removing subscriptions to RSS feeds, converting subscriptions to PDF files, and displaying the Tracker window within Acrobat DC. This section describes URL commands that can do the following tasks:

- Add a subscription
- Update a subscription
- Select a subscription
- Remove a subscription
- Convert a subscription to a PDF file
- Display Tracker

Note: For all of the Tracker URL commands, Acrobat DC is launched if it is not already open.

2.1.1 Add a subscription

```
acrobat:Inbox?addFeed=<URL>
```

Adds the RSS feed specified by `URL`, and opens the Tracker window. A security dialog box may appear, indicating the source of the subscription and providing the user with the option to cancel the feed subscription.

2.1.2 Update a subscription

```
acrobat:Inbox?updateFeed=<URL>
```

Asynchronously updates the RSS feed specified by `URL`, and opens the Tracker window.

2.1.3 Select a subscription

```
acrobat:Inbox?selectFeed=<URL>
```

Selects the RSS feed specified by `URL`, and opens the Tracker window.

Alternatively, the following notation can be used to select an item within a feed:

```
acrobat:Inbox?selectFeed=<URL>#<GUID>
```

In this case, `GUID` is the RSS GUID item. For example, if the user subscribes to the RSS feed at http://example.org/RSS/example and the item within the feed has a GUID of http://example.org/RSS/abc, the following notation is used:

```
acrobat:Inbox?selectFeed=<http://example.org/RSS/example>#

  <GUID>@<http://example.org/RSS/abc>
```

2.1.4 Remove a subscription

```
acrobat:Inbox?removeFeed=<URL>
```

Removes the RSS subscription specified by `URL`. A verification dialog box appears, providing the user with the option to cancel the subscription removal.

2.1.5 Convert a subscription to a PDF file

```
acrobat:Inbox?convert=<URL>
```

Converts the RSS subscription specified by `URL` to a PDF file. The user is not required to subscribe to the feed, and in such cases the feed does not remain in Tracker after the conversion has taken place.

2.1.6 Display Tracker

```
acrobat:Inbox?show
```

Displays the Tracker window.

2.2 RSS XML feed extensions

You can add XML extensions to RSS 2.0 feeds to customize the user interface for the subscriptions in Tracker. The extensions are based on the following specifications:

- Namespace
- Channel extensions
- Item extensions

2.2.1 Namespace

Tracker extensions use the namespace defined at http://ns.adobe.com/Acrobat/RSS/Inbox to extend RSS 2.0. You can set the namespace prefix by adding the following XML attribute to the RSS node:

```
xmlns:inbox="http://ns.adobe.com/Acrobat/RSS/Inbox"
```

For more information on XML namespaces, see http://www.w3.org/TR/REC-xml-names/ .

2.2.2 Channel extensions

You can add extensions as children of the RSS 2.0 `<channel/>` element. These can be used to perform the following actions:

- Customizing the user interface
- Grouping items within a feed
- Sorting items within a feed
- Assigning icons to feeds
- Marking feed items as unread

Customizing the user interface

```
<inbox:feedUI/>
```

Provides a URL for a JavaScript user interface driver. For security reasons, the URL for the driver must be relative to the URL of the XML document containing the RSS feed. For example, the following code uses `myCustomUI.js`:

```
<inbox:feedUI>myCustomUI.js</inbox:feedUI>
```

Note: This type of URL is known as a code subscription feed . Code subscription is done through the same mechanism as RSS and the contents are stored locally. However, the files are standard JavaScript files. Code subscription feeds are checked for updates infrequently, and update only the local copy since code changes do not occur dynamically. The code changes do not take effect until the next time Tracker is started.

Grouping items within a feed

```
<inbox:groupBy/>

  <inbox:grouping/>
```

Provides the name of an element that should be used for grouping items within a feed. For example, `author` would indicate that the RSS `<author/>` element should be used for grouping items according to author name, as shown in the following code:

```
<inbox:groupBy>author</inbox:groupBy>
```

If the grouping element is an XML namespace, the elements can be specified using the following notation: `namespace:local name`. For example, if the namespace is http://example.org/myInfoNamespace/ and the local name is `timeZone`, the notation would appear as follows:

```
<inbox:groupBy>http://example.org/myInfoNamespace/:timeZone</
↪inbox:groupBy>
```

The `<inbox:groupBy/>` element can use multiple child `<inbox:grouping/>` elements. This allows for multiple levels of grouping for feed items. In the following example, items will be grouped according to author name, and within those groups there will be subgroups of items grouped according to the year:

```
<inbox:groupBy>
     <grouping>author</grouping>
```

(continues on next page)

(continued from previous page)

```
    <grouping>year</grouping>
</inbox:groupBy>
```

Sorting items within a feed

```
<inbox:sortBy/>
```

Provides the name of an element to be used for lexically sorting items within a feed. For example, `author` would indicate that the RSS `author` element should be used for sorting items according to author name, as shown in the following code:

```
<inbox:sortBy>author</inbox:sortBy>
```

If the grouping element is an XML namespace, the elements can be specified using the following notation: `namespace:local name`. For example, if the namespace is http://example.org/myInfoNamespace/ and the local name is `timeZone`, the notation would appear as follows:

```
<inbox:sortBy>http://example.org/myInfoNamespace/:timeZone</
↪inbox:sortBy>
```

Assigning icons to feeds

```
<inbox:icon/>
```

Provides the URL of an icon to be used for the tree view item icon for a feed. For security reasons, the URL for the icon must be relative to the URL of the XML document containing the RSS feed. You can subscribe to icons provided they are in PNG format and are less than 20 by 20 pixels in size. For example, the following code assigns the icon contained in `Circle.png` to the feed:

```
<inbox:icon>Circle.png</inbox:icon>
```

Marking feed items as unread

```
<inbox:markUnread/>
```

Indicates whether unread items in the feed should be highlighted. The default is for the items to be marked. The following code indicates that unread items should be marked:

```
<inbox:markUnread>true</inbox:markUnread>
```

2.2.3 Item extensions

You can add extensions as children of the RSS 2.0 `channel` element. These can be used to perform the following actions:

- Nesting feed items
- Hiding feed items

As with channel extensions, item extensions can also be used to perform the following actions:

- Customizing the user interface
- Assigning icons to feeds

Nesting feed items

```
<inbox:subfeed/>
```

Provides a URL to an RSS feed to be nested under the current Tracker item. This allows a multi-level hierarchy of information to be displayed in Tracker. For security reasons, the URL for the nested feed must be relative to the URL of the XML document containing the current RSS feed.

Hiding feed items

```
<inbox:hidden/>
```

Indicates that the item should not be displayed in Tracker. This can be used to force the display of empty groups.

2.3 User interface driver

The Tracker extensions provide the `inbox:feedUI` element, which consists of a URL to a code subscription feed. If no user interface driver is specified, a generic driver is used. If only some of the methods are implemented in the user interface driver object defined in the JavaScript file, the default RSS driver is used to implement the remaining methods.

The user interface driver JavaScript file should return a driver object as the event result, as shown in the following example:

```
var ui = { /* User interface driver object */ };
 event.result = ui;
```

The user interface driver is active when a feed or any of its elements are selected. Multiple feeds can use the same driver.

The user interface driver requires the following object definitions:

- Dialog object
- Driver object
- Layout description object
- RSS object
- Selection object

2.3.1 Dialog object

The `dialog` object is described in the JavaScript for Acrobat API Reference . However, when using Tracker, it is augmented with the following methods:

- getSelection
- isFeedSelection
- isItemSelection
- isGroupSelection
- getFeed

getSelection

This method returns the current `selection` object. This can be a subscription feed or a group or item within a feed, and it can be identified using the `isFeedSelection`, `isGroupSelection`, and `isItemSelection` methods.

isFeedSelection

This method returns `true` if the current `selection` object is a subscription feed as described in the `RSS.getContents` method.

isItemSelection

This method returns `true` if the current `selection` object is an item within a subscription feed.

isGroupSelection

This method returns `true` if the current `selection` object is a string representing the name of a group within a subscription feed.

getFeed

This method returns the subscription feed of the current selection.

2.3.2 Driver object

The JavaScript driver feed is a script that is executed. The script should set the `event.result` property to an object implementing some or all of the methods defined in this section.

createToolBar

This method displays a toolbar at the top of Tracker when items that use the driver are selected. The method should return an array of objects (one for each toolbar button), each containing the following properties:

Property	Type	Description
cType	String	• `button`: A toolbar button. • `separator`: A toolbar button separator bar. • `textButton`: A button with a text field.
cName	String	A unique name for the button.
cDisplayName	String	A string to display on the button's label.
cTip	String	A tooltip to display for the button.
onExecute	Function	A function that is executed when the button is clicked. The function is passed a `selection` object as a parameter. If `cType` is `textButton`, there will be a second parameter containing the value of the text field.
onEnabled	Function	A function that is executed to determine if the button should be enabled. The function is passed a `selection` object as a parameter, and returns `true` if the button should be enabled, `false` otherwise. If the method is not defined, the button is always enabled.

createContext

This method displays a window at the bottom of Tracker when items that use the driver are selected. The method returns a `layout` description object (see Layout description object for more information). If the driver does not provide this method, the default behavior is for the HTML content in the RSS feed to be rendered in the context window.

getInitiateName

This method provides a user interface name for an initiation workflow provided by the `getInitiateMenu` method.

getInitiateMenu

This method provides a menu item in the context menu of the Acrobat DC Send for Review menu. This allows a user interface driver to provide a workflow initiation user interface. The method returns an array of objects corresponding to menu items, each having the following properties:

Property	Type	Description
cName	String	A unique name for the menu.
cDisplayName	String	A localized display name for the menu item.
onExecute	Function	A function that is executed when the menu is selected. The function is passed a `selection` object as a parameter.
onEnabled	Function	A function that is executed to determine if the menu should be enabled. The function is passed a `selection` object as a parameter, and returns `true` if the menu should be enabled, `false` otherwise. If the method is not defined, the menu is always enabled.
onMarked	Function	A function that is executed to determine if the menu should be marked. The function is passed a `selection` object as a parameter, and returns `true` if the menu should be marked, `false` otherwise. If the method is not defined, the menu is always unmarked.
bSeparator	Boolean	Determines whether this menu item is a separator.
oSubMenu	Array	An array of objects containing the same properties as the menu items for a hierarchical sub menu.

getFeedContextMenu

This method is passed a `selection` object as a parameter, and returns an array of objects corresponding to menu items in a context menu, each having the following properties:

Property	Type	Description
cName	String	A unique name for the menu.
cDisplayName	String	A localized display name for the menu item.
onExecute	Function	A function that is executed when the menu is selected. The function is passed a `selection` object as a parameter.
onEnabled	Function	A function that is executed to determine if the menu should be enabled. The function is passed a `selection` object as a parameter, and returns `true` if the menu should be enabled, `false` otherwise. If the method is not defined, the menu is always enabled.
onMarked	Function	A function that is executed to determine if the menu should be marked. The function is passed a `selection` object as a parameter, and returns `true` if the menu should be marked, `false` otherwise. If the method is not defined, the menu is always unmarked.
bSeparator	Boolean	Determines whether this menu item is a separator.
oSubMenu	Array	An array of objects containing the same properties as the menu items for a hierarchical sub menu.

getItemContextMenu

This method is passed a feed and feed item as parameters, and returns an array of objects corresponding to menu items, each having the following properties:

Property	Type	Description
cName	String	A unique name for the menu.
cDisplayName	String	A localized display name for the menu item.
onExecute	Function	A function that is executed when the menu is selected. The function is passed a `selection` object as a parameter.
onEnabled	Function	A function that is executed to determine if the menu should be enabled. The function is passed a `selection` object as a parameter, and returns `true` if the menu should be enabled, `false` otherwise. If the method is not defined, the menu is always enabled.
onMarked	Function	A function that is executed to determine if the menu should be marked. The function is passed a `selection` object as a parameter, and returns `true` if the menu should be marked, `false` otherwise. If the method is not defined, the menu is always unmarked.
bSeparator	Boolean	Determines whether this menu item is a separator.
oSubMenu	Array	An array of objects containing the same properties as the menu items for a hierarchical sub menu.

getFeedTitle

This method is passed a `selection` object as a parameter, and returns a display string for the feed title.

getItemTitle

This method is passed a `selection` object as a parameter, and returns a display string for the feed item text.

getGroupTitle

This method is passed a `selection` object as a parameter, and returns a display string for the text used for a group.

getFeedTip

This method is passed a `selection` object as a parameter, and returns a display string for the feed tooltip.

getGroupTip

This method is passed a `selection` object as a parameter, and returns a display string for the tooltip used for a group.

onSelectFeed

This method is passed a `selection` object as a parameter, and is called when a feed is selected.

onSelectItem

This method is passed a `selection` object as a parameter, and is called when a feed item is selected.

onActivateFeed

This method is passed a `selection` object as a parameter, and is called when a feed is double-clicked.

onActivateItem

This method is passed a `selection` object as a parameter, and is called when a feed item is double-clicked.

onActivateGroup

This method is passed a `selection` object as a parameter, and is called when a group is double-clicked.

canDeleteFeed

This method is passed a `selection` object as a parameter, and is called to determine if a feed can be deleted. If the feed can be deleted, the method returns `true`.

onDeleteFeed

This method is passed a `selection` object as a parameter, and is called when a feed is deleted. If the method returns `true`, the feed has been deleted. In this case, the method should modify the subscription accordingly.

canDeleteItem

This method is passed a `selection` object as a parameter, and is called to determine if an item can be deleted. If the item can be deleted, the method returns `true`.

onDeleteItem

This method is passed a `selection` object as a parameter, and is called when an item is deleted. If the method returns `true`, the item has been deleted. In this case, the method should modify the subscription accordingly.

canDeleteGroup

This method is passed a `selection` object as a parameter, and is called to determine if a group can be deleted. If the feed can be deleted, the method returns `true`.

onDeleteGroup

This method is passed a `selection` object as a parameter, and is called when a group is deleted. If the method returns `true`, the group has been deleted. In this case, the method should modify the subscription accordingly.

shouldSort

This method determines whether the feed items should be sorted. If it returns `true`, the items will be sorted.

sortCompare

This method is used to sort feed items. It is passed a feed description and two item descriptions `A` and `B` as parameters, and returns one of the following values:

- If `A` < `B`, the method returns `-1`.
- If `A` = `B`, the method returns `0`.
- If `A` > `B`, the method returns `1`.

shouldGroup

This method determines whether the feed items should be grouped together. If it returns `true`, the items will be grouped.

groupItem

This method is used to group the items in a feed. It is passed an item description as a parameter, and returns either a localized category name for the method or an empty string indicating that there is no grouping category.

shouldFilterFeed

This method determines whether an entire feed should be displayed. It is passed a feed description and the document ID of the current document as parameters. If it returns `false`, the feed is hidden, otherwise it is displayed.

shouldFilterItem

This method determines whether a feed item should be displayed. It is passed an item description as a parameter. If it returns `false`, the feed is hidden, otherwise it is displayed.

getIdlePeriod

This method returns the period in seconds that the `idle` method should be called while the user interface for this driver is active and displayed.

idle

This method is passed the periodicity, obtained from the `getIdlePeriod` method, as a parameter, assuming that the periodicity was a positive value. It should run for the number of seconds specified by the periodicity.

getHTMLRendition

This method returns an HTML representation of the `selection` object. The HTML representation is typically used to convert the content to a PDF file using the WebCapture plug-in.

canChangeGroups

This method indicates whether the user interface driver supports drag and drop operations on items between groups in the feed. The method returns `true` if the operations are supported, `false` otherwise, and its outcome is based on the values returned by the `canModifyItemGroup` and `modifyItemGroup` methods.

canModifyItemGroup

This method determines whether the user can perform a drag and drop operation on an item within a group. It receives three parameters: a `selection` object for the currently selected item, an array of the group labels to which the item currently belongs, and an array of destination group labels. The method returns `true` if the operation is allowed.

modifyItemGroup

This method is called when the user performs a drag and drop operation on an item within a group. It receives three parameters: a `selection` object for the currently selected item, an array of the group labels to which the item currently belongs, and an array of destination group labels. The method returns `true` if the operation is allowed.

2.3.3 Layout description object

The layout of Tracker's context window is described by a `layout` description object. This object contains a hierarchical set of views and a set of event handlers for those views. Each view has a view identifier (its `item_id` property) that provides access to the view value as well as a notification callback. The `layout` description object is returned by the user interface driver object's `createContext` method.

Property	Type	Description
name	String	A generic label that may be used for a variety of view types, such as a `button` or `cluster`.
type	String	• `view`: A generic group of other views (default). • `cluster`: A labeled cluster of views. • `check_box`: A check box. • `ok`: An OK button. • `ok_cancel`: An OK/Cancel button group. • `ok_cancel_other`: An OK/Cancel/Other button group. • `button`: A button. • `html`: An HTML view.
align_children	String	• `align_center`: Centers the element within its parent. • `align_fill`: Stretches the element to fill the entire row. • `align_left`: Aligns the element to the left within its parent. • `align_right`: Aligns the element to the right within its parent.
align	Array	• `align_center`: Centers the children. • `align_fill`: Distributes the children to fill the entire row. • `align_left`: Aligns the children to the left. • `align_right`: Aligns the children to the right.
elements	Array	An array of child views for this view.
width	Number	The view's width in points.
height	Number	The view's height in points.
item_id	String	The view identifier used to get and set the view value and to set a handler method.

2.3.4 RSS object

The `RSS` object is used to add, remove, and update RSS feeds, and manage their content and user interface behavior.

Property	Type	Access	Description
feeds	Array	R	This property contains the currently subscribed feed URLs.
hasExternalReader	Boolean	R	This property indicates that there is an external news reader available that can h • addFeed • removeFeed • getContents • update • addUI

addFeed

Adds the feed specified by `cURL` to the subscriptions managed by Acrobat DC.

Parameters

cURL	The URL of the RSS feed.
bHidden	Optional. Determines whether the RSS feed is displayed in Tracker. If `true` (default), the feed will not be displayed.
bPersistent	Optional. Determines whether the RSS feed contents will be stored persistently and will be available when Acrobat DC is offline or restarted. If `true` (default), the feed contents will be stored persistently.
cType	Optional. The type of RSS feed to be added. The possible values are: `RSS`: The subscription is an XML document (RSS 0.9 or higher, 1.0, or 2.0). `JS`: The subscription is a JavaScript document. `Icon`: The subscription is a PNG icon. `UI`: The subscription is a JavaScript document to be registered as a UI driver.
bUpdateAsync	Optional. Determines whether the subscription contents are initially retrieved asynchronously. If `true` (default), the contents are retrieved asynchronously. If `false`, the contents will be retrieved before the method returns, which may take a significant period of time and should be avoided if possible.

removeFeed

Removes the subscription specified by `cURL` that was previously added by calling `addFeed`.

Parameters

cURL	The URL of the RSS feed.

getContents

Returns the current contents of the RSS cache for the feed specified by `cURL`.

Parameters

cURL	The URL of the RSS feed.

Returns

`Object`

If the feed has not yet been fetched, the result is a `null` object. Otherwise, the object returned depends on the type of feed. If the subscription is an RSS or ATOM feed, it returns an object with the following properties.

Property	Description
Title	The title of the RSS feed.
Link	A link to an HTML document describing the RSS feed.
Description	A description of the RSS feed. This may be an HTML fragment.
ModDate	The last time the RSS feed was modified.
Extensions	An object describing any extensions made to the RSS feed. The `key` of each object property will be the name of the element. If there is more than one occurrence of an element, the `value` will be an array of those element values. Otherwise the `value` will be the contents of the element.
Item	An array of objects corresponding to the Items in the RSS feed. The Items have the following properties: `Description`: The description of the item. This may be an HTML fragment. `Title`: The title of the item. `Link`: A hyperlink to a related document. `ModDate`: The modification time of the item. `Extensions`: An object describing extensions made to the RSS Item. The `key` of each object property is the name of the element. If there is more than one occurrence of an element, the `value` is an array of those element values. Otherwise the `value` is the contents of the element.

update

Forces the update of the subscription specified by `cURL`.

Parameters

cURL	Optional. The URL of the RSS feed. If no URL is specified, all RSS feeds will be updated.
bForce	Optional. Determines whether the RSS feed is updated even if it is not required. If `true` (default), the feed is updated. If `false`, the feed is only updated when required. Updates are normally based on the last modification, time to live (TTL), and skipdays or skiphours feed properties.
bAsync	Optional. Determines whether the subscription contents are updated asynchronously. If `true` (default), the contents are updated asynchronously. If `false`, the contents are updated synchronously.
cType	Optional. Determines the type of update: `local`: Updates the user interface without refreshing the subscription content. `subscription` (default): Updates only the subscription contents. `recursive`: Updates the subscription and any related contents, including the user interface, related subscriptions, and icons.

addUI

Adds a user interface driver for a feed.

Parameters

cURL	The URL that RSS feeds can reference to request the driver. The convention is that machine local resources use `local://<uniqueName>` for their URLs.
oDriver	A `UI Driver` object. See User interface driver for more information on specifying this parameter.

2.3.5 Selection object

The `selection` object is passed to many of the user interface driver methods and indicates the selection context for the method.

Property	Type	Description
type	String	Indicates whether a feed, a feed item, or a group of items is currently selected. The following values may be used: • `feed`: The selection is a top level feed. • `item`: The selection is an item in a feed. • `group`: The selection is a group of items.
feed	Object	A description of the feed. For more information, see the RSS object's `getContents` method.
item	Object	If the selection is an item in a feed, the property will be the item selected. For more information, see the RSS object's `getContents` method. The property is undefined if the selection is not a feed item.
group	String	If the selection is a group of items, the property will be the name of the selected group. The property is undefined if the selection is not a group of items.

CHAPTER

THREE

CUSTOMIZATION EXAMPLES

This chapter provides two examples of customizing the Tracker user interface. To implement these examples, place the code in a directory that is published by a web server.

Note: In both examples, it is assumed that the example code is located at http://example.org/path.

3.1 Grouping elements

You can group elements by customizing an RSS feed in an RSS 2.0 XML document that uses the `inbox:groupBy` element. In this case, the grouping is accomplished through the creation of a `timeZone` element defined in the XML namespace http://example.org/myInfoNamespace/, using the following syntax:

```
<inbox:groupBy>http://example.org/myInfoNamespace/:timeZone</
↪inbox:groupBy>
```

In this RSS document, the `<rss>` tag defines the XML namespace used in this sample for `timeZone` and assigns it to the namespace prefix `info` , as shown in the following code:

```
<rss version="2.0" xmlns:inbox="http://ns.adobe.com/Acrobat/RSS/Inbox/"
 xmlns:info="http://example.org/myInfoNamespace/">
```

This makes it possible for each RSS feed item to include an `info:timeZone` tag and use it to assign each item to the correct group. For example, the following element is included in one of the items so that it is assigned to the group for Central Standard Time:

```
<info:timeZone>Central Standard Time</info:timeZone>
```

Tracker uses this metadata to perform its grouping operations, while other RSS aggregators ignore the information. You can experiment with this by changing the following RSS document so that Tracker performs the grouping using the `info:state` tag. You can do this by changing the `inbox:groupBy` element.

```
<rss version="2.0" xmlns:inbox="http://ns.adobe.com/Acrobat/RSS/Inbox/"
xmlns:info="http://example.org/myInfoNamespace/">
   <channel>
       <title>Example1</title>
       <link>http://example.org/Example1</link>
       <description>Example of how to extend Tracker with
           Tags</description>
       <language>en-gb</language>

       <!-- Refresh once every 10 minutes -->
       <ttl>10</ttl>

       <!-- Group the elements by our custom metadata -->
       <inbox:groupBy>http://example.org/myInfoNamespace/:timeZone</
↪inbox:groupBy>

       <!-- Set an icon for the feed using a relative URL -->
       <inbox:icon>Circle.png</inbox:icon>

       <item>
           <title>Chicago</title>
           <info:timeZone>Central Standard Time</info:timeZone>
           <info:state>IL</info:state>

           <!-- Set an icon for the item using a relative URL -->
           <inbox:icon>Square.png</inbox:icon>

           <link>http://example.org/Chicago</link>
           <guid>http://example.org/Chicago</guid>
           <description>The Windy City</description>
       </item>

       <item>
           <title>San Francisco</title>
           <info:timeZone>Pacific Standard Time</info:timeZone>
           <info:state>CA</info:state>

           <!-- Set an icon for the item using a relative URL -->
           <inbox:icon>Square.png</inbox:icon>

           <link>http://example.org/SF</link>
           <guid>http://example.org/SF</guid>
           <description>Home of the Golden Gate Bridge</description>
       </item>

       <item>
```

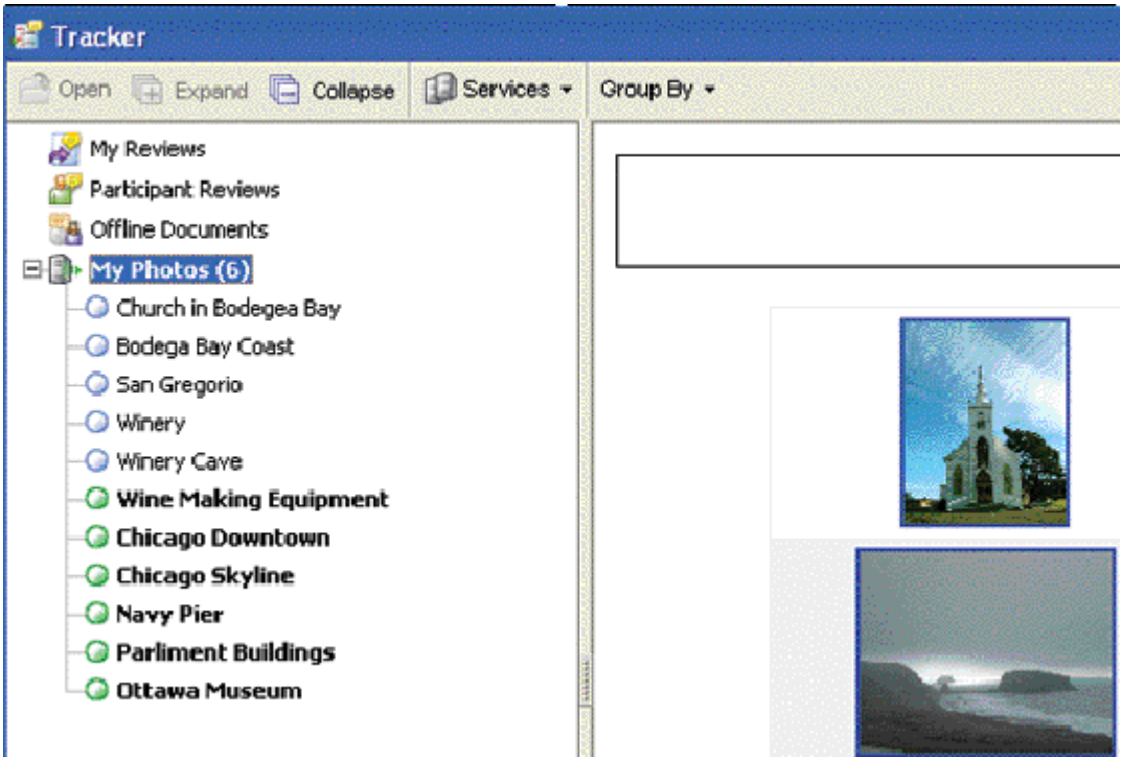
(continues on next page)

(continued from previous page)

```
            <title>New York</title>
            <info:timeZone>Eastern Standard Time</info:timeZone>
            <info:state>NY York</info:state>

            <!-- Set an icon for the item using a relative URL -->
            <inbox:icon>Square.png</inbox:icon>

            <link>http://example.org/NY</link>
            <guid>http://example.org/NY</guid>
            <description>The Big Apple</description>
        </item>

        <item>
            <title>Boston</title>
            <info:timeZone>Eastern Standard Time</info:timeZone>
            <info:state>MA</info:state>

            <!-- Set an icon for the item using a relative URL -->
            <inbox:icon>Square.png</inbox:icon>

            <link>http://example.org/Boston</link>
            <guid>http://example.org/Boston</guid>
            <description>Bean Town</description>
        </item>

        <item>
            <title>Philadelphia</title>
            <info:timeZone>Eastern Standard Time</info:timeZone>
            <info:state>PA</info:state>

            <!-- Set an icon for the item using a relative URL -->
            <inbox:icon>Square.png</inbox:icon>

            <link>http://example.org/Philadelphia</link>
            <guid>http://example.org/PhiladelphiaNY</guid>
            <description>Cheese Steaks</description>
        </item>

    </channel>
</rss>
```


3.2 Adding an external interface driver

The following graphic shows a Group By button added to the user interface through the use of an external JavaScript driver.



The driver is referenced with the `inbox:feedUI` tag in the RSS document, loaded from the server, and stored persistently on the client while it is needed. The driver provides a toolbar button that allows dynamic grouping of photos within the user interface.

The RSS document introduces an XML namespace called `photo`, which is used for the custom metadata:

```
<rss xmlns:inbox="http://ns.adobe.com/Acrobat/RSS/Inbox/"
xmlns:photo="http://example.org/Photos/" version="2.0">
```

This provides the metadata used by the user interface driver. For example, each item contains custom metadata for event and location, as shown in the following code:

```
<photo:event>Day Trip</photo:event>
<photo:location>Bodega Bay</photo:location>
```

To bind the RSS document to a JavaScript user interface driver, use the `inbox:feedUI` tag with a relative URL. In this case, the driver file is in the same URL location as the RSS file:

```
<inbox:feedUI>Example2.js</inbox:feedUI>
```

The user interface driver consists of the following methods:

- `createToolBar`
- `shouldGroup`
- `groupItem`
- `createContext`

The `createToolBar` method creates the toolbar by returning an array object containing one element for each toolbar button. In this case, it adds the Group By button, which has a menu allowing the user to group items according to the following choices: None, Location, or Event. The Location and Event options correspond to the `photo:location` and `photo:event` tags in the RSS document:

```
aFileTypes:
 [
     ["None", ""],
     ["Location", "http://example.org/Photos/:location"],
     ["Event", "http://example.org/Photos/:event"]
 ],
```

The user's choice is stored in the `cGroup` member variable, and the method forces the RSS feed to update by invoking the `RSS.update` method:

```
var val = this.aFileTypes[i];
 ...
 cValue: val[1],
 onExecute: function(selection)
 {
     this.oUI.cGroup = this.cValue;
     RSS.update(selection.feed.URL, true);
 },
```

The `shouldGroup` method determines whether grouping should apply to the current selection by examining the `cGroup` member variable. In this case, if the user chose None then the string assigned to `cGroup` would have a length of zero:

```
shouldGroup: function(selection)
 {
     return this.cGroup.length != 0;
 },
```

The `groupItem` method determines which group should be applied to the `selection` object, which it receives as a parameter. It does this by returning the value of the metadata element, which it retrieves through the `item.Extension` object:

```
groupItem: function(selection)
 {
     return selection.item.Extension[this.cGroup];
 },
```

The `createContext` method generates the right portion of the user interface based on the current selection. The layout is a simple HTML view, so creating the context view is accomplished by generating HTML according to the current selection. The method returns a JavaScript object containing the following methods and properties:

- `initialize`
- `onSelectionChanged`
- `showFeedView`
- `showItemView`
- `description`

These are described below.

The `initialize` method is called when the driver is initialized and stores a copy of the `dialog` object for future use.

The `onSelectionChanged` method is called when a feed is selected. The selection may be the root level for the current subscription (the feed), an item, or a group. This method determines the selection type and calls the corresponding method for the type of item selected.

The `showFeedView` method generates an HTML string (`content`) that is used to produce a table containing thumbnails for each photo. If a group is selected, it displays only those feed items belonging to the group. Once the HTML string is created, it is loaded into the HTML view, which has a view identifier of `"html"` .

```
var stm = SOAP.streamFromString(content);
 dialog.load({"html": stm});
```

The `showItemView` method is similar to the `showFeedView` method, but is restricted to the current item.

The `description` property contains the view description. In this case it contains generic wrapper view elements that control alignment and an HTML view. This is where all the content is generated when the selection changes (using `dialog.load`).

The RSS document and user interface driver code are as follows:

```
<?xml version="1.0"?>
<rss xmlns:inbox="http://ns.adobe.com/Acrobat/RSS/Inbox/"
xmlns:photo="http://example.org/Photos/" version="2.0">
   <channel>
```

(continues on next page)

(continued from previous page)

```
        <title>My Photos</title>
        <link>http://example.org/</link>
        <description>My Photos</description>

        <!-- Add a UI driver for this feed -->
        <inbox:feedUI>Example2.js</inbox:feedUI>

        <ttl>60</ttl>
        <item>
            <title>Church in Bodegea Bay</title>
            <link>bodega bay church.jpg</link>
            <description>&lt;a href="bodega bay church.jpg"&gt;&lt;
                img width=500
                src="bodega bay church.jpg"&gt;&lt;/a&gt;
            </description>
            <guid>bodega bay church.jpg/PTO.pdf</guid>

            <!-- Custom Metadata in our namespace - event and location -
↪->
            <photo:event>Day Trip</photo:event>
            <photo:location>Bodega Bay</photo:location>

            <pubDate>Sun, 22 Dec 2002 17:49:45 GMT</pubDate>
        </item>
        <item>
            <title>Bodega Bay Coast</title>
            <link>IM000853.JPG</link>
            <description>&lt;a href="IM000853.JPG"&gt;&lt;
                img width=500 src="IM000853.JPG"&gt;&lt;/a&gt;
            </description>
            <guid>IM000853.JPG</guid>
            <photo:event>Day Trip</photo:event>
            <photo:location>Bodega Bay</photo:location>
            <pubDate>Wed, 01 Jan 2003 17:49:45 GMT</pubDate>
        </item>
        <item>
            <title>San Gregorio</title>
            <link>IM002002_edited.JPG</link>
            <description>&lt;a href="IM002002_edited.JPG"&gt;&lt;
                img width=500 src="IM002002_edited.JPG"
                &gt;&lt;/a&gt;
            </description>
            <guid>IM002002_edited.JPG</guid>
            <photo:event>Day Trip</photo:event>
            <photo:location>San Gregorio</photo:location>
```

(continues on next page)

(continued from previous page)

```
            <pubDate>Wed, 09 Jul 2003 17:49:45 GMT</pubDate>
        </item>
        <item>
            <title>Winery</title>
            <link>IM002265_edited.JPG</link>
            <description>&lt;a href="IM002265_edited.JPG"&gt;&lt;
                img width=500 src="IM002265_edited.JPG"
                &gt;&lt;/a&gt;
            </description>
            <guid>IM002265_edited.JPG/PTO.pdf</guid>
            <photo:event>Day Trip</photo:event>
            <photo:location>Napa</photo:location>
            <pubDate>Wed, 09 Jul 2003 17:49:45 GMT</pubDate>
        </item>
        <item>
            <title>Winery Cave</title>
            <link>IM002285.JPG</link>
            <description>&lt;a href="IM002285.JPG"&gt;&lt;img width=500
                src="IM002285.JPG"&gt;&lt;/a&gt;
            </description>
            <guid>IM002285.JPG</guid>
            <photo:event>Day Trip</photo:event>
            <photo:location>Napa</photo:location>
            <pubDate>Wed, 09 Jul 2003 17:49:45 GMT</pubDate>
        </item>
        <item>
            <title>Wine Making Equipment</title>
            <link>IM002293.JPG</link>
            <description>&lt;a href="IM002293.JPG"&gt;&lt;img width=500
                src="IM002293.JPG"&gt;&lt;/a&gt;
            </description>
            <guid>IM002293.JPG</guid>
            <photo:event>Day Trip</photo:event>
            <photo:location>Napa</photo:location>
            <pubDate>Wed, 09 Jul 2003 17:49:45 GMT</pubDate>
        </item>
        <item>
            <title>Chicago Downtown</title>
            <link>IM002311.JPG</link>
            <description>&lt;a href="IM002311.JPG"&gt;&lt;img width=500
                src="IM002311.JPG"&gt;&lt;/a&gt;
            </description>
            <guid>IM002311.JPG</guid>
            <photo:event>Christmas</photo:event>
            <photo:location>Chicago</photo:location>
```

(continues on next page)

(continued from previous page)

```
        <pubDate>Wed, 09 Jul 2003 17:49:45 GMT</pubDate>
    </item>
    <item>
        <title>Chicago Skyline</title>
        <link>IM002313.JPG</link>
        <description>&lt;a href="IM002313.JPG"&gt;&lt;img width=500
            src="IM002313.JPG"&gt;&lt;/a&gt;
        </description>
        <guid>IM002313.JPG</guid>
        <photo:event>Christmas</photo:event>
        <photo:location>Chicago</photo:location>
        <pubDate>Wed, 09 Jul 2003 17:49:45 GMT</pubDate>
    </item>
    <item>
        <title>Navy Pier</title>
        <link>IM002327.JPG</link>
        <description>&lt;a href="IM002327.JPG"&gt;&lt;
            img width=500 src="IM002327.JPG"&gt;&lt;/a&gt;
        </description>
        <guid>IM002327.JPG</guid>
        <photo:event>Christmas</photo:event>
        <photo:location>Chicago</photo:location>
        <pubDate>Wed, 09 Jul 2003 17:49:45 GMT</pubDate>
    </item>
    <item>
        <title>Parliment Buildings</title>
        <link>IM002583.JPG</link>
        <description>&lt;a href="IM002583.JPG"&gt;&lt;img width=500
            src="IM002583.JPG"&gt;&lt;/a&gt;
        </description>
        <guid>IM002583.JPG</guid>
        <photo:event>Christmas</photo:event>
        <photo:location>Ottawa</photo:location>
        <pubDate>Wed, 09 Jul 2003 17:49:45 GMT</pubDate>
    </item>
    <item>
        <title>Ottawa Museum</title>
        <link>IM002613.JPG</link>
        <description>&lt;a href="IIM002613.JPG"&gt;&lt;img width=500
            src="IM002613.JPG"&gt;&lt;/a&gt;
        </description>
        <guid>IM002613.JPG</guid>
        <photo:event>Christmas</photo:event>
        <photo:location>Ottawa</photo:location>
        <pubDate>Wed, 09 Jul 2003 17:49:45 GMT</pubDate>
```

(continues on next page)

(continued from previous page)

```
        </item>
    </channel>
</rss>

function CreateExample2UIObject()
{
var ui =
{
    cGroup: "",
    instance: null,
    createToolBar: function()
    {
        var result =
        [
            {
                oUI: ui,
                cType: "button",
                cName: "org.example.example2.groupBy",
                cDisplayName: "Group By",
                cTip: "Group Photos by the selected property",
                aFileTypes:
                [
                    ["None", ""],
                    ["Location", "http://example.org/Photos/:location"],
                    ["Event", "http://example.org/Photos/:event"]
                ],

                getMenu: function(selection)
                {
                    var result = new Array;
                    for(var i in this.aFileTypes)
                    {
                        var val = this.aFileTypes[i];

                        result[result.length] =
                        {
                            oUI: this.oUI,
                            cName: val[0],
                            cDisplayName: val[0],
                            cValue: val[1],
                            onExecute: function(selection)
                            {
                                this.oUI.cGroup = this.cValue;
                                RSS.update(selection.feed.URL, true);
                            },
```

(continues on next page)

(continued from previous page)

```
                        onMarked: function(selection)
                        {
                            return this.oUI.cGroup == this.cValue;
                        }
                    };
                }

                return result;
            }
        }
    ];

    return result;
},

shouldGroup: function(selection)
{
    return this.cGroup.length != 0;
},

groupItem: function(selection)
{
    return selection.item.Extension[this.cGroup];
},

createContext: function()
{
    this.context =
    {
        driver: ui,
        initialize: function(dialog)
        {
            ui.instance = dialog;
        },

        onSelectionChanged: function(dialog)
        {
            if(dialog.isItemSelection())
                this.showItemView(dialog);
            else if(
                        dialog.isFeedSelection() ||
                        dialog.isGroupSelection()
            )
                this.showFeedView(dialog);
```

(continues on next page)

(continued from previous page)

```
        },

        showFeedView: function(dialog)
        {
            var feed = dialog.getFeed();
            var extension = feed.Extension;

            var content = "<HTML><HEAD><STYLE TYPE="text/css">.
                evenBkg {background-color: #EDF3FE; }.
                oddBkg {background-color: #ffffff; }.
                titleBkg { background-color: #EEEEEE; }.
                column { padding: 5px; }</STYLE></HEAD><BODY>";

            content += "<BASE href="" + feed.URL + "">";


            content += "<DIV style="font-family:sans-serif;
                font-size:120%;
                padding: 5px;
                border: 1px solid #000000;
                text-align: center;">";

            content += "<B>" + feed.Title + "</B><BR>";

            if(dialog.isGroupSelection())
                content += dialog.getSelection();

            content += "<BR></DIV>";
            content += "<BR>";
            content += "<CENTER>";

            var items = feed.Items;

            if(typeof items != "undefined" && items.length > 0)
            {
                content += "<TABLE align=center cellspacing=0
                    cellpadding=5 style="width: 75%;
                    border: 1px solid #EEEEEE;
                    text-align:center">";

                for(var i in items)
                {
                    var item = items[i];
if(item.Extension["http://ns.adobe.com/Acrobat/RSS/Inbox/:hidden"])
                        continue;
```

(continues on next page)

(continued from previous page)

```
                        if(dialog.isGroupSelection())
                        {
  if(item.Extension[this.driver.cGroup] != dialog.getSelection())
                                continue;
                        }

                        var background = "oddBkg";
                        if(highlight) background = "evenBkg";

                        var link = "<A STYLE="color: #000000;
                            text-decoration: none" href="" +
                            item.Link + ""/>";
                        content += "<TR CLASS="" + background + "">";
                        content += "<TD>" + link + "<IMG height=100 src=
↪"" +
                            item.Link + ""></A></TD>";
                        content += "<TD>" + link;
                        content += item.Title
                        content += "</A></TD>";
                        content += "</TR>";
                        highlight = !highlight;
                    }

                    content += "</TABLE>";
                }

                content = content + "</BODY></HTML>";
                var stm = SOAP.streamFromString(content);
                dialog.load({"html": stm});
            },

            showItemView: function(dialog)
            {
                if(!dialog.isItemSelection()) return;

                var selection = dialog.getSelection();

                var content = "<html><body>";
                content += "<BASE href="" + dialog.getFeed().URL + "">";
                content += "<center><B>" + selection.Title + "</B><BR>";
                content += selection.Description + "</body></html>";

                var stm = SOAP.streamFromString(content);
                dialog.load({"html": stm});
```

(continues on next page)

(continued from previous page)

```
            },

            // The dialog box description
            description:
            {
                name: "Panel",
                align: "align_center",
                align_children: "align_fill",
                elements:        // Child element array
                [
                    {
                        type: "view",
                        elements:        // Child element array
                        [
                            {
                                type: "html_view",
                                item_id: "html"
                            }
                        ]
                    }
                ]
            }
        };
        return this.context;
    }
};
return ui;
}
event.result = CreateExample2UIObject();
```

