ACCESS NetFront™ Browser Widgets

Marcin Hanclik, R&D, ACCESS CO., LTD.

NetFront Browser Widgets (hereafter NFBW) is a widget solution for mobile terminals and many other Internet enabled appliances. NFBW conforms to the standard W3C specification, and also includes original extensions. NFBW operates on top of the NetFront Browser library.

1. Background information

ACCESS is the worldwide supplier of the software solutions for mobile devices and Internet appliances. One of the most widely recognized and used ACCESS products is the NetFront Browser. As of November 2008 a total of 1552 various models and 639 million devices embedding the NetFront product have been shipped, equating to 6.3 units per second delivered to market during recent years.

2. NetFront Browser Widgets Features

NFBW is based on W3C Widgets 1.0 specification. The W3C Widgets 1.0 working draft does not currently include some important parts, such as the Security Model. Since W3C Widgets 1.0 mainly targets the PC environment, on which hardware resources are abundant, environments with limited hardware resources such as mobile terminals are not fully considered. Therefore, the NFBW specification has been prepared by extending and complementing the W3C Widgets 1.0 specification. ACCESS also actively participates in OMTP BONDI working groups. NFBW supports the XML Signature as a signature method for the widget content.

2.1 Using Terminal Information from JavaScript

In NFBW, it is possible to access terminal hardware information or data stored on the terminal by extending JavaScript functions. With extended JavaScript functions, NFBW users can utilize widgets with features and functionality that cannot be implemented using HTML alone.

The JavaScript extensions and corresponding stub functions can be generated from the IDL files, thus providing a convenient way of extending the APIs available to the widget developers.

2.2 Security Model Based on PKI

NFBW uses a Security Model based on PKI (Public Key Infrastructure) to ensure security for executing widgets. NFBW Security Model does not allow unauthenticated widgets to access personal information on the terminal. By only permitting connections with specific servers, NFBW Security Model prevents widgets from loading malicious JavaScript code.

2.3 Considerations for the Mobile Terminal Environment

NFBW can provide services to environments with limited hardware resources such as mobile terminals as well as environments with abundant hardware resources. NFBW provides implementers of Widget Player Applications and widget content creators with a mechanism for displaying multiple widgets within the small display area of the terminal at the same time and functions for reducing the power consumed by the terminal.

3. Security Model

3.1 Same Origin Policy

NFBW uses Same Origin Policy that prevents the transfer of information between widgets from different origins. Origins are identified using the “domain” the widget belongs to that is composed of the following three elements.

- URI scheme
- Host name
- Port number

The Browser Application also applies the Same Origin Policy to XMLHttpRequest. On the other hand, the Widget Player Application that complies with the NFBW specification does not apply the Same Origin Policy to XMLHttpRequest. This enables the widget content creator to utilize the Web API provided by the server of domains other than the one widget belongs to via XMLHttpRequest.

Although the NFBW relaxes the Same Origin Policy for XMLHttpRequest, another mechanism is provided to restrict network access.

3.2 Network Access Restrictions

The NFBW-based Widget Player Application can put restrictions on the general network access. Restriction set using the `<access>` element regulates the generic network access and `<netaccess>` element of `config.xml` limits the servers – host and port pairs - the widget can connect to.

3.3 Access Restrictions on JavaScript Object

The NFBW Security Model is based on the security model provided in the Java Mobile Information Device Profile 2.0 (hereafter MIDP 2.0). There are differences between Java and JavaScript in terms of the language specification, and this Security Model contains some NFBW-specific specifications.

The following MIDP 2.0 terms are used also to describe the NFBW Security Model:

• *Protection domain:*	Logical unit for defining access permission for each function group.
• *Function group:*	Group of JavaScript object properties to which access restrictions are applied, or definitions of functions such as file access.
• *Permission:*	Access permission setting for function group.

4. Policy Definition File

In NFBW the policy definition file is prepared on the assumption that NFBW extended JavaScript objects and certificates may be dynamically added to the terminal. It is possible to apply security settings for new JavaScript properties and certificates on the terminal by updating the policy definition file when they are dynamically added. Only one policy definition file is active at any given time.

The policy definition file follows the XML syntax as specified below.

Element Name/ Attribute Name		Definition	Description
`security` element		Mandatory	Root element of policy file.
`ProtectionDomain` element		Mandatory	Defines protection domain. This element must always have a `certificate` element as a child element.
	`name` attribute	Mandatory	Protection domain name. Used when referencing with `policy` element.
	`identifier` attribute	Mandatory	Integer, 1 or higher, that represents a protection domain. Used by NFBW integration (peer) code to identify a protection domain. Must not be duplicated in the entire policy file.
`certificate` element		Mandatory	Specifies base64-encoded root certificate.
	`name` attribute	Mandatory	Certificate name. Used in the Security Manager.
`functionGroup` element		Mandatory	Defines a function group. This element can have a `JavaScript` element as a child element.
	`name` attribute	Mandatory	Name of function group. Used when referencing with `policy` element.
	`permission` attribute	Mandatory	Default permission of function group. Specify `allow, session, oneshot` or `prohibit`.
`JavaScript` element		Optional	Defines an extended JavaScript object or property. This element has no child elements.
	`class` attribute	Mandatory	Class name of extended JavaScript object.
	`property` attribute	Optional	Property name of extended JavaScript object.
`policy` element		Optional	Defines function groups that belong to a protection domain. This element must always have a `functionGroup` element as a child element. The `permission` attribute permissions specified in the `functionGroup` element of this element's child elements overwrite the default permissions for Widget Content authenticated by certification of the `protectionDomain` attribute of this element.
	`protectionDomain` attribute	Mandatory	Protection domain that sets function group. Use the name specified by the `name` attribute of the `protectionDomain` element.

The access, execution of method, and property acquisition of extended JavaScript objects are restricted in the Security Manager. If a prohibited access, method, or property acquisition of JavaScript object is executed, a `SecurityException` exception will occur.

4.1 Protection Domain

A protection domain is a logical unit that is used to define access permission settings for a function group. Each protection domain is associated with different certificates, and the protection domain is identified by the certificate. Multiple certificates can be associated with one protection domain. One certificate cannot be associated with multiple protection domains. It is possible to define a special protection domain that is not associated with any certificate.

The protection domain to which widget content belongs is determined by the certificate chain associated with the widget content. The protection domain can be set only in relation to root certificates of the certificate chain.

The default permission is applied to widget content that is not associated with a certificate chain, that is, unsigned content or widget content that is associated with a certificate chain that is not associated with a protection domain.

4.2 Function Group

A function group defines a group of JavaScript object properties to be protected. Access permission is set by each function group.

4.3 Permission

Permissions are the access restrictions for a function group. The permission may have one of four values: “allow”, “session”, “oneshot” or “prohibit”.

If one of the following conditions is met when accessing a function group, the JavaScript execution environment raises a `SecurityException` exception in the following cases:

- permission’s value is “prohibit”
- permission’s value is “session” or “oneshot”, and rejected as a result of user inquiry.

A policy definition file may look as follows:

```
<security>
 <protectionDomain name=”DomainA” identifier=’1’>
   <certificate name=”cert1”><!--Certificate encoded in BASE64--></certificate>
 </protectionDomain>

 <protectionDomain name=”DomainB” identifier=’2’>
   <certificate name=”cert1”><!--Certificate encoded in BASE64--></certificate>
 </protectionDomain>

 <functionGroup name=”FunctionA” permission=”allow”>
   <javascript class=”System” />
   <javascript class=”Power” property=”charging” />
 </functionGroup>

 <functionGroup name=”FunctionB” permission=”prohibit”>
   <javascript class=”Power” property=”remaining” />
 </functionGroup>
 <!-- Default permissions of function group B -->

 <functionGroup name=”RTSP” permission=”prohibit”/>
 <!-- Default permissions of function group RTSP -->

 <functionGroup name=”File Scheme” permission=”prohibit”/>

 <policy protectionDomain=”DomainA”>
  <!-- Overwrite default permission of function group B -->
  <functionGroup name=”FunctionA” permission=”allow” />
 </policy>

 <policy protectionDomain=”DomainB”>
  <functionGroup name=”FunctionA” permission=”allow” />
  <functionGroup name=”FunctionB” permission=”session” />
  <functionGroup name=”RTSP” permission=”allow” />
  <functionGroup name=”File Scheme” permission=”oneshot” />
 </policy>

</security>
```

5. Differences with the Browser

A Widget Player Application conforming to the NFBW specification displays and executes content that is written in HTML, CSS, and JavaScript, as in the case of the Browser Application. However, the Widget Player Application behaves differently in some ways from the Browser Application, because it displays and executes specific content - the widgets. Behavior differences between the Widget Player Application and the Browser Application are described below.

5.1 History Function

The Browser Application manages content URIs visited by the user and the order in which URIs are visited. The Browser Application provides functions for displaying the list of visited URIs and for returning to the last visited page using these URIs. On the other hand, the Widget Player Application does not use the concept of "visits" since it aims at executing specific widget content. The following functions related to the visit history are disabled in the Widget Player Application.

- Coloring visited links using `vlink` attribute of `<body>` element
- Changing style of visited links using `:visited` pseudo class of CSS
- Part of properties of JavaScript's Window object and History object
- Functions that can be operated from the UI, such as displaying visit history in a list and returning to the last visited page.

5.2 JavaScript Same Origin Policy

Considering the fact that widget content uses Web APIs provided by external servers in different domains, NFBW has made modifications to the Same Origin Policy as highlighted in the previous section.

5.3 Restrictions on Servers That Can Be Communicated With

NFBW may restrict servers that can be communicated with from widgets by using the <netaccess> element in the config.xml as specified earlier.

5.4 Handling HTTP Cookie

If the cookie property of the HTMLDocument object is referenced by an HTML document within the widget package, the Widget Player Application returns an empty string. Also, if a value is set for the cookie property of the HTMLDocument object, the Widget Player Application does not save the HTTP Cookie. This is because the URIs of the HTML documents in the widget package are neither HTTP URIs nor HTTPS URIs, but Widget URIs. HTTP Cookie is accessible only from the HTMLDocument object of an HTTP URI or HTTPS URI. If the cookie property of the HTMLDocument object is accessed in an HTML document obtained from an external server, the Widget Player Application behaves in the same way as the Browser Application.

5.5 Handling Referer

If the referer is a Widget URI, the Widget Player Application handles the referer as follows:

- HTTP Referer header is not set
- An empty string is returned as the referer property of the HTMLDocument object

5.6 Dialog Display Using JavaScript

NFBW restricts the display of dialog boxes that use the `alert()` method, the `prompt()` method, or the `confirm()` method in JavaScript in order to avoid the following issues:

- Dialog box is continuously displayed by a loop
- Because it is not possible to identify which widget displays a dialog box, users may be confused.

5.7 Widget URI

NFBW defines a special URI for accessing content within a widget package. This is called the Widget URI.

5.8 Page Transition Behavior

Page transition behavior in the Widget Player Application differs from that of the Browser Application. The variations in behavior take into account whether the referred URI is the internal or external one and whether the page transition is related to anchor selection, form submission or rewriting of the `location.href` or `src` attributes.

5.9 Transparent Window Support

The Widget Player Application supports transparent windows without a window frame and title bar to enable widget content of any shape to be displayed. The Widget Player Application displays widget content through which the background image (e.g. standby screen wallpaper on a mobile phone, etc.) can be seen by drawing a PNG image, etc. specifying transparency on the transparent window.

The Widget Player Application treats `<iframe>` and `<frame>` elements as completely opaque based on the specification of NetFront Browser that is used as the rendering engine for HTML documents. The initial value for the background colors of `<iframe>` and `<frame>` elements is white, and the frame is always drawn.

Whether the Widget Player Application supports transparent windows depends on the terminal environment in which the Widget Player Application is executed. Transparent windows may not be supported or the gradation of transparency may be restricted depending on the terminal's capabilities.

5.10 Display states

NFBW defines three types of display states so the widget display size can be changed according to the display area size on the terminal.

Normal Display State (Float State)

The normal display state is used to position more than one widget on the standby screen, etc. Widgets on the standby screen can be aligned, overlapped, or moved to any position. For this reason, the normal display state is sometimes called the float state. When the state switches from another display state to the normal display state, the `onrestore` event handler of the widget object is called. The widget content creator can switch the display content to the normal display state using the `onrestore` event handler.

The width and height for the normal display state are specified in the `<widget>` element or `<display>` element of `config.xml` using the `width` and `height` attributes.

Maximized Display State

In the maximized display state, a widget is displayed within a wider area than the normal display state. The number of widgets that can be displayed at the same time in this state depends on the terminal. For example, it is possible to make a specification that allows only one widget to be displayed in this state for an environment with a small display area such as a mobile terminal, and allows more than one widget to be displayed concurrently within an environment with a large display area such as a TV.

When the state switches from another display state to the maximized display state, the `onmaximize` event handler of the Widget object is called. The widget content creator can implement widget content so that the maximum display state can display detailed information that cannot be displayed in the normal display state using the `onmaximize` event handler.

The width and height of the maximized display state can be specified in the `<xwidget>` element or `<display>` element in `config.xml` using the `maximizedwidth` and `maximizedheight` attributes. If the `maximizedwidth` and `maximizedheight` attributes are not specified, the width and height of the maximized display state will be the same as the terminal's display area. If the sizes specified for the `maximizedwidth` attribute and `maximizedheight` attribute are larger than the terminal's display area, the behavior depends on the particular terminal implementation.

Docked Display State

Docked display state displays more than one widget on the standby screen, etc. using small rectangular windows. This state is used to show the minimum amount of information desired to be updated regularly when displaying a widget. When the state switches from another display state to the docked display state, the `ondock` event handler of the Widget object is called. The widget content creator can switch the display content to that of docked display state using the `ondock` event handler.

The width and height of the docked display state depend on the terminal. The recommended size is approximately 32 x 32 to 40 x 40 pixels for QVGA (320 x 240), and 64 x 64 to 80 x 80 pixels for VGA (640 x 480), but they may differ depending on the terminal. The widget content creator must consider that the size may change to a certain degree in the docked display state.

6. More …

ACCESS NetFront Browser Widgets Empower SOFTBANK MOBILE (press release)
http://www.access-company.com/news/press/ACCESS/2008/20081030_softbank.html

More information can be found at:
http://www.access.co.jp/nfbwc/en/

ACCESS™

http://www.access-company.com