TIZEN™

Tizen Web Application Development for Beginners

Version 1.0 (2014/09/XX)

Table of Contents

About This Document

This document is aimed at Tizen application developers to understand the Tizen platform and develop Tizen applications. The information in this document is based on the Tizen 2.3 alpha release at the time of writing.

For more details and up-to-date reference information, go to the Tizen Developer site for the Tizen SDK Help documentation.

The information in this document is still subject to change. See the official Tizen SDK documentation for the final version.

1.Introduction to Tizen

Understanding Tizen

Tizen is an open-source operating system targeted for multiple device categories, such as smart phones, in-vehicle infotainment (IVI) devices, wearable devices, smart TVs, computers, cameras, and printers.

Tizen has the following key characteristics:

Tizen as an Open-source Software Platform

Tizen is a flexible open-source operating system built from the ground up to meet the needs of all parties of the mobile and connected device ecosystem, including device manufacturers, mobile network operators, application developers, and independent software vendors. As an open-source software platform, Tizen provides many opportunities to both individuals and companies.

Tizen is developed by a community under open-source governance, and is open to all who wish to participate. Developers can contribute to the Tizen project at different levels, for example, as platform contributors or application developers.

Tizen as a Web Software Platform

Tizen's main focus for application development is HTML5, which is currently considered the preferred development environment for mobile applications. HTML5 allows developers to build cross-platform applications: to write code once and run it on multiple platforms.

Tizen supports most of the official HTML5 standards as well as certain supplementary and legacy standards, and most of the major W3C APIs.

The Tizen Web engine contains a rendering engine, which combines markup content (such as HTML and image files) with formatting information (such as CSS) and displays the formatted content on the screen, and a JavaScript engine, which interprets and executes JavaScript.

Tizen as an Industry-supported Software Platform

The Tizen Association provides strong industry support to the Tizen project. The Tizen Association was formed to guide Tizen's industry role, including the gathering of requirements, identification and facilitation of service models, and overall industry marketing and education. For more information, see www.tizenassociation.org.

The Tizen project resides within the Linux Foundation and is governed by a Technical Steering Group. The Technical Steering Group is the primary decision-making body for the Tizen project, with a focus on platform development and delivery as well as on the formation of working groups to support device verticals.

Tizen Community Web Sites

The main Tizen community Web sites are:

- **Tizen main site** (www.tizen.org)

  The main site of the Tizen project features articles and blogs. You can create a Tizen account for other Tizen sites and get the latest news about the project and events.

- **Tizen Developers** (developer.tizen.org)

  The official site for Tizen developers provides the Tizen SDK and various resources for developing Tizen applications. You can discuss technical issues with other developers in the discussion forums and find helpful sample codes for developing your applications.

- **Tizen Source** (source.tizen.org)

  The official site for the Tizen open-source project provides the Tizen source code and instructions for building it. You can use Git to track source code releases and retrieve the latest revisions.

- **Tizen Project JIRA** (bugs.tizen.org/jira)

  The official JIRA site for the Tizen project allows you to register, track bugs, and make suggestions for new features.

- **Tizen Wiki** (wiki.tizen.org)

  The Tizen wiki allows you to collaborate on the documentation for the Tizen project.

2. Tizen Architecture

The Tizen architecture includes both Web APIs for developing Web applications and native APIs for developing native applications. Since Web applications form the main approach to developing for Tizen, this document focuses only on the Web APIs.

The following figure shows the Tizen architecture.

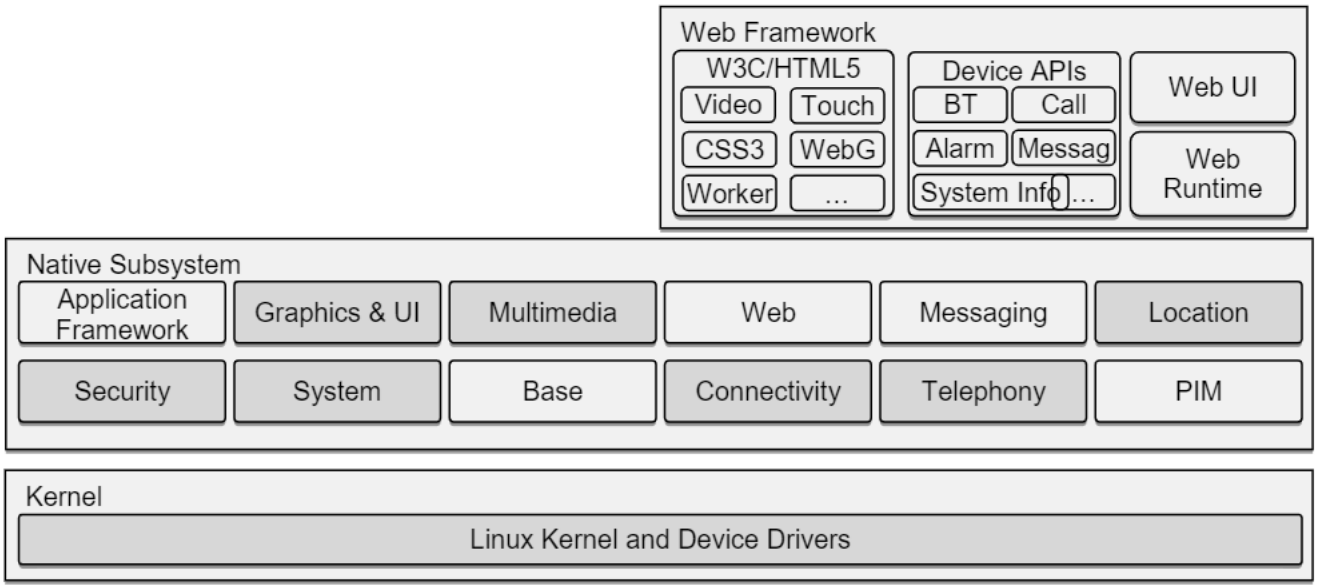

Figure 1: Tizen architecture

The Tizen architecture consists of the following subsystems:

- **Web framework**

  The Web framework implements the latest Web technologies, accessible through the W3C/HTML5 APIs. The framework supports a large number of HTML5 features defined by W3C and other standardization groups, such as audio, video, forms, 2D canvas elements, vibration, Web Graphics Library (WebGL), CSS3, WebSocket, and web workers.

  The framework also includes Device APIs, which allow you to access device functions, such as alarms, wireless data exchange, and messaging. The framework provides the device functions through a strict rule-based security control system that prevents the malicious use of the APIs.

  The Core Web module provides a complete implementation of the Web framework, optimized for low-power devices. In addition to the Web APIs, the implementation includes WebKit, a layout engine for rendering Web pages in browsers, as well as a Web runtime for running Web applications.

- **Native subsystem**

  The Native subsystem provides features required by the Web framework. The subsystem consists of open-source libraries and an additional set of native APIs, which the Web framework uses. For more information about the native APIs, see the Tizen SDK Help.

- **Kernel**

  The Kernel subsystem contains the Linux kernel, customized for Tizen, and device drivers.

3.Introduction to Tizen Web APIs

You develop Tizen Web applications with the Tizen Web APIs, which consist of W3C/HTML5 APIs, supplementary Web APIs, and Tizen Device APIs.

The following figure shows the different types of Web APIs.

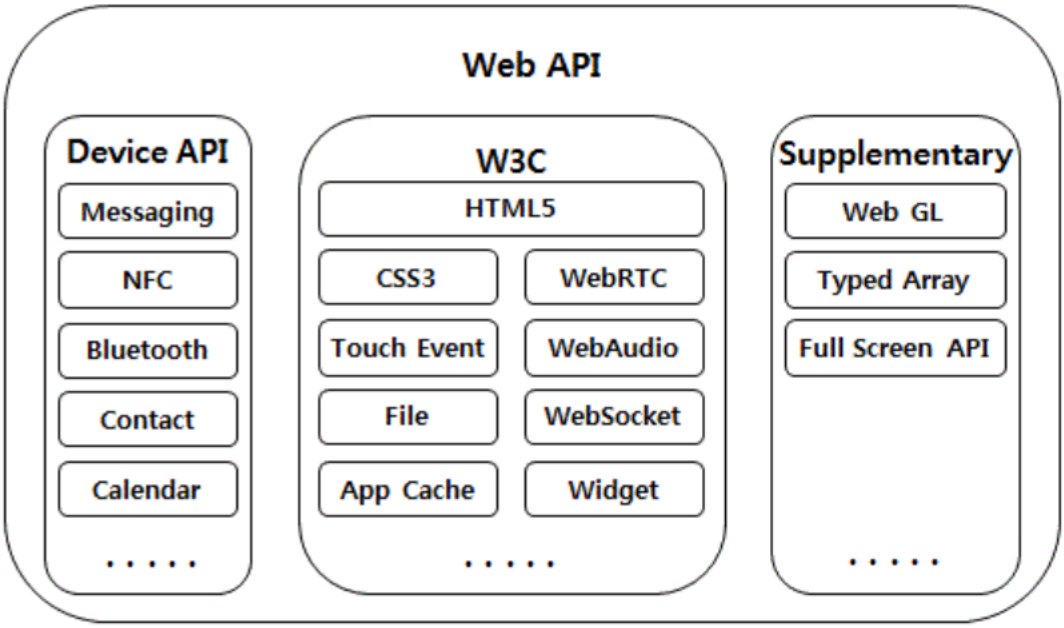

Figure 2: Tizen Web APIs

Tizen Device APIs

The Device APIs are based on JavaScript and provide advanced access to the device's platform capabilities.

The main Device APIs are:

- **Alarm API**

  The Alarm API allows you to schedule system alarms. Alarms enable you to run other applications to perform specific operations at specific times. You can schedule an alarm to be triggered once or repeatedly at predefined intervals.

- **Application API**

  The Application API allows you to retrieve information about the applications installed or currently running on the device. You can also receive notifications when applications are installed, updated, or removed, and perform application management tasks, such as launching or exiting an application.

- **Bluetooth API**

  The Bluetooth API allows you to use the device's Bluetooth features, such as managing the local Bluetooth adapter, pairing the device with other Bluetooth devices, and exchanging data over a Bluetooth connection.

- **Calendar API**

  The Calendar API allows you to manage events and tasks in a calendar. A calendar is a collection of events or tasks, depending on the calendar type. Each event or task has a series of attributes, such as purpose, starting time, and duration.

- **Contact API**

  The Contact API allows you to manage the contacts and people listed in an address book. A contact object is always associated with a specific address book. A person object is an aggregation of one or more contacts associated with the same person.

- **Content API**

  The Content API allows you to search for content such as images, videos, and music stored locally on the device. You can also perform content management tasks, such as viewing and updating content attributes.

- **Download API**

  The Download API allows you to download files from the Internet. You can also monitor the download progress and status.

- **Filesystem API**

  The Filesystem API allows you to manage the files and folders of the device file system. The Filesystem API provides access to the non-restricted portions of the file system, which are represented as virtual root locations. The virtual roots form a collection of locations that together function as a single virtual file system, which your application can access.

- **Messaging API**

  The Messaging API allows you to use messaging functions for SMS, MMS, and email communication. The HTML5 messaging process uses uniform resource identifiers (URIs).

- **NFC API**

  The NFC API allows you to use the Near Field Communication (NFC) service for exchanging data between NFC devices ("peers") or tags. The devices can share contacts, photos, and videos, and can also act as smart cards. You can use an NFC device to communicate with NFC tags for a range of activities, such as paying bills or downloading coupons.

- **Notification API**

  The Notification API allows you to provide notifications about application events.

- **Package API**

  The Package API allows you to retrieve detailed information about packages, such as name, ID, icon path, and version. You can also receive notifications when packages are installed, updated, or removed, and perform package management tasks, such as installing or uninstalling packages.

- **Power API**

  The Power API allows you to access the device's power resources. Currently, the screen and CPU power resources are supported, allowing you to request a specific power state and control the brightness of the screen.

- **System Information API**

  The System Information API allows you to access the device's system properties. You can retrieve various device and system details, such as the current battery level, amount of available storage, and state of the cellular network connection.

- **System Setting API**

  The System Setting API allows you to access the device's settings for the home and lock screen wallpapers, incoming call ringtone, and email notification tone.

- **Time API**

  The Time API allows you to use locale-specific calendar features by retrieving date and time information. You can also change the date, time, and time zone of the device, and perform calculations related to date and time.

- **Tizen API**

  The Tizen API allows you to use common Tizen functions. The Tizen API contains filters and sorting modes for query methods, generic success and error event handlers, a generic error interface, and a simple coordinate interface for defining location information.

- **Web Setting API**

  The Web Setting API allows you to set Web view properties.

For more information about the Device APIs, see the Tizen SDK Help.

4. Getting Started with Web Application Development

Tizen provides you with the tools to manage your application's life-cycle from conception and design, through development and release, to end-of-life application retirement.

To develop a Tizen Web application:

1. Plan and design the application
2. Install the Tizen SDK
3. Create an application project
4. Create the application UI
5. Build the application
6. Run and debug the application
7. Package the application
8. Localize the application, if necessary

Planning and Designing the Application

The first step in developing a Tizen Web application is planning and designing the application using the design tools of your choice.

After you have finished planning and designing your application, install the Tizen SDK and create the application project.

Installing the Tizen SDK

The Tizen SDK is a comprehensive set of tools for developing Tizen Web and native applications. It consists of an IDE, emulator, toolchain, sample code, and help documentation.

When installing the SDK, you can choose either online installation or offline installation.

Prerequisites

Before starting the installation, check that your computer meets the following system requirements:

- Operating system:
  - Mac OS® X 10.7 Lion (64-bit) or Mac OS® X 10.8 Mountain Lion (64-bit) or Mac OS® X 10.9 Mavericks (64-bit)
  - Microsoft Windows® 7 (32-bit or 64-bit) or Microsoft Windows® 8 (32-bit or 64-bit)
  - Ubuntu® 12.04 or 12.10 or 14.04 (32-bit or 64-bit)
- Dual-core 2 GHz CPU
- 2 GB RAM
- 6 GB free disk space
- Local administrator privileges

For more information about the prerequisites, see https://developer.tizen.org/downloads/sdk/installing-sdk/prerequisites-tizen-sdk.

Installing the Tizen SDK (Online)

To install the Tizen SDK online:

1. Download the SDK Install Manager.
2. Run the SDK Install Manager.
3. Click **Next**.
4. Accept the terms and conditions, and click **Next**.
5. Select the components you want to install, and click **Next**.
6. Select the SDK home folder, and click **Install**.

The SDK setup wizard informs you when the installation is complete. You can now create the application project in the IDE.

Installing the Tizen SDK (Offline)

To install the SDK offline from an image file:

1. Download the SDK Install Manager.
2. Download the SDK image file.
3. Run the SDK Install Manager.
4. Click **Advanced**.
5. In the **Advanced Configuration** window, select **SDK Image**.
6. Click the folder button, browse to the SDK image file, and click **OK**.
7. Click **Next**.
8. Accept the terms and conditions, and click **Next**.
9. Select the components you want to install, and click **Next**.
10. Select the SDK home folder, and click **Install**.

The SDK setup wizard informs you when the installation is complete. You can now create the application project in the IDE.

Creating the Application Project

After you have planned and designed your application, and installed the SDK, you are ready to create a Web application project in the IDE.

When creating the application project, use an applicable project template or sample. Based on the template or sample, the Project Wizard automatically implements the basic functionalities the application needs to run. You can select from a variety of templates and samples. You can also create your own user templates.

Finally, to achieve the required functionality and features, set the project properties and configure the application carefully.

Creating a Web Application Project

To create a Web application project for your application:

1. In the IDE, select **File > New > Tizen Web Project**.

   If the project option you want is not visible, make sure you are using the Tizen Web perspective:

   a. Select **Window > Open Perspective > Other**.

   b. Select the **Tizen Web** perspective.

2. In the **New Tizen Web Project** window, define the project details:

   a. Select the template or sample on which you want to base the application.

   b. Define a name for the project.

> **Note:**
>
> The Tizen API names cannot be used as project names. The project name must be between 2 and 49 characters in length, and can only contain the following alphanumeric characters: a-z, A-Z, 0-9.

3. If enabled, click the **Next** button, and customize the jQuery Mobile or the Tizen Web UI Framework template settings.
4. Click **Finish**.

The new application project is shown in the **Project Explorer** view, with default content in the `config.xml` file as well as in several project folders.

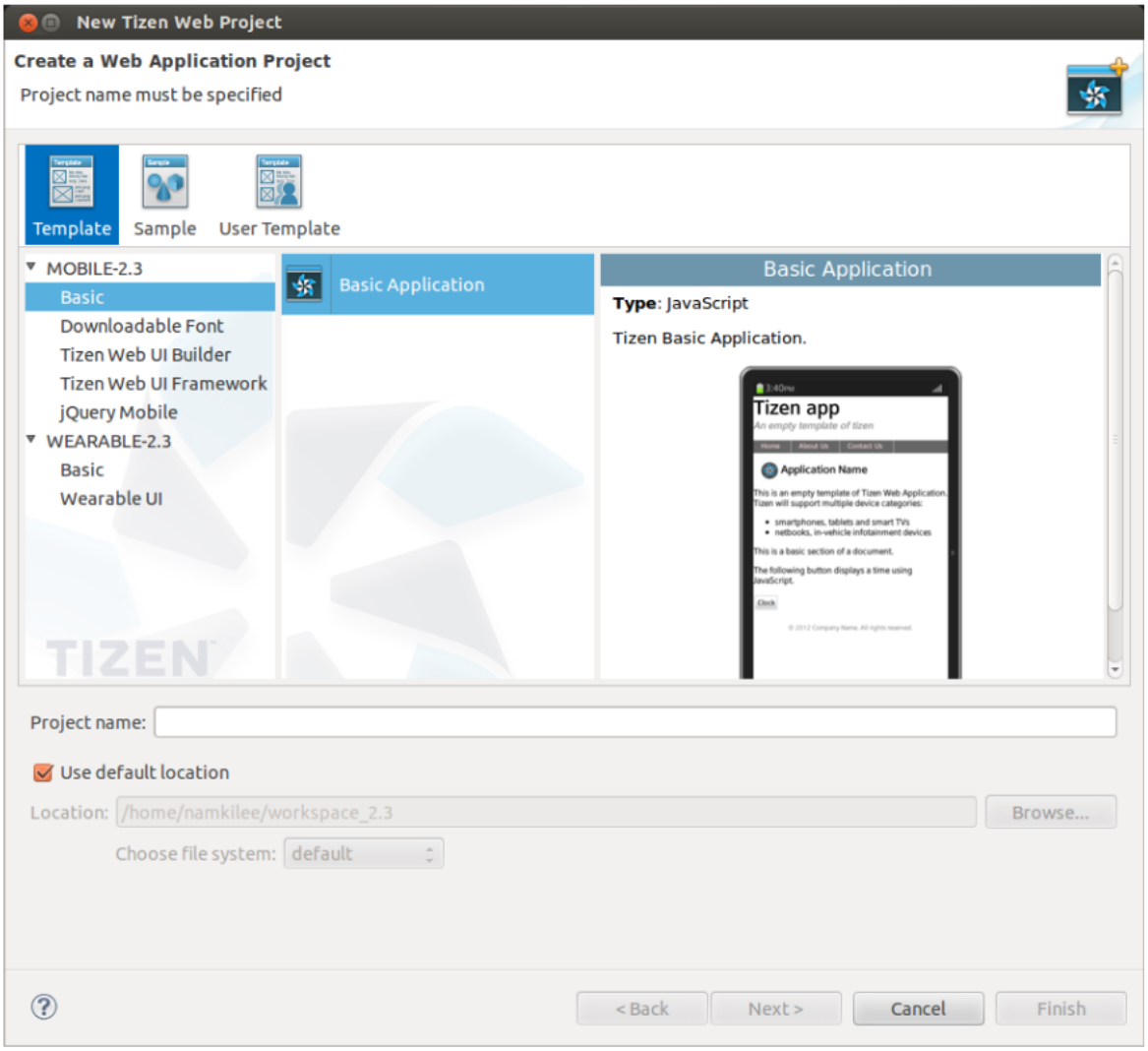

Figure 3: Creating a Tizen Web application project

Importing a Project

To import an existing Tizen application project into the IDE:

1. In the IDE, select **File > Import > Tizen > Tizen Web Projects and Widget file**.
2. Click **Browse**, and select the root folder containing your existing project or widget file (`.wgt`).

   After selecting the project, you can copy it into the workspace by selecting **Copy projects into workspace**.
3. Click **Finish**.

Supported Templates and Samples

You can create Tizen Web applications based on a variety of templates and samples. In addition to the predefined templates, you can also create your own user templates.

The following table lists the predefined templates available to Tizen Web application development.

Table 1: Predefined templates

Project type	Application type	Description
Basic	Blank Application	This is a blank Tizen application template.
Tizen Web UI Builder	• Empty Application • Single Page Application • Multi Page Application • Navigation Application	These templates are based on the Tizen Web UI Builder.
Tizen Web UI Framework	• Single Page Application • Multi Page Application • Master Detail Application • Navigation Application	These templates are based on the Tizen Web UI Framework.
jQuery Mobile	• Single Page Application • Multi Page Application • Master Detail Application • Navigation Application	These templates are based on jQuery Mobile. You can change the application page header, body, and footer based on the following color themes: • A: Black • B: Blue • C: Grey • D: Light grey • E: Yellow

Creating a User Template

You can create your own user templates either by copying existing user templates or by exporting existing projects as user templates. You can then use these templates to create new projects.

The user templates are located in the `<TIZEN_SDK_DATA>/ide/user-templates/web` folder and in user-defined folders. The exact location of the `<TIZEN_SDK_DATA>` folder depends on the operating system:

- Linux/Mac OS® X: `/home/${USER_NAME}/tizen-sdk-data`
- Windows: `C:\tizen-sdk-data`

The project folder of a user template contains project-related files that are copied to new projects that are created from the template. You can modify these files in your new project.

The following table describes the files included for a user template in addition to the project folder content.

Table 2: User template files

File	Contents
`description.xml`	This file consists of the following template description information that is displayed

	in the Project Wizard: • `<SampleName>`: Project type name. Several project types can be created using the same user template. • `<SampleVersion>`: Project type version. • `<Preview>`: Preview file name. • `<Description>`: Description title.
`tizen-app-template.xml`	This file consists of the following template definition information that is displayed in the Project Wizard: • `<template-name>`: User template name. • `<widget-type>`: Widget type. Set to `TIZEN` by default. • `<description-file-name>`: Description file name. Set to `description.xml` by default. • icon64 and icon32 attributes: Icon image file names. The number suffixes refer to the icon size (in pixels).

Copying a User Template

To create a new user template from an existing user template:

1. Copy and rename the corresponding `<TIZEN_SDK_DATA>/ide/user-templates/web` folder.

   The folder contains a default template layout.
2. Edit the files in the new folder, or add new files, such as snapshot or icon images and CSS and JavaScript files, as needed.

Exporting a Project as a User Template

To export an existing project as a new user template:

1. In the IDE, select **File > Export > Tizen > User Template**.
2. In the **User Template Export Wizard** window, define the project and user template details:
   a. Select the source project.
   b. Define a name for the user template.
   c. Click **Browse**, and select the export location for the user template.

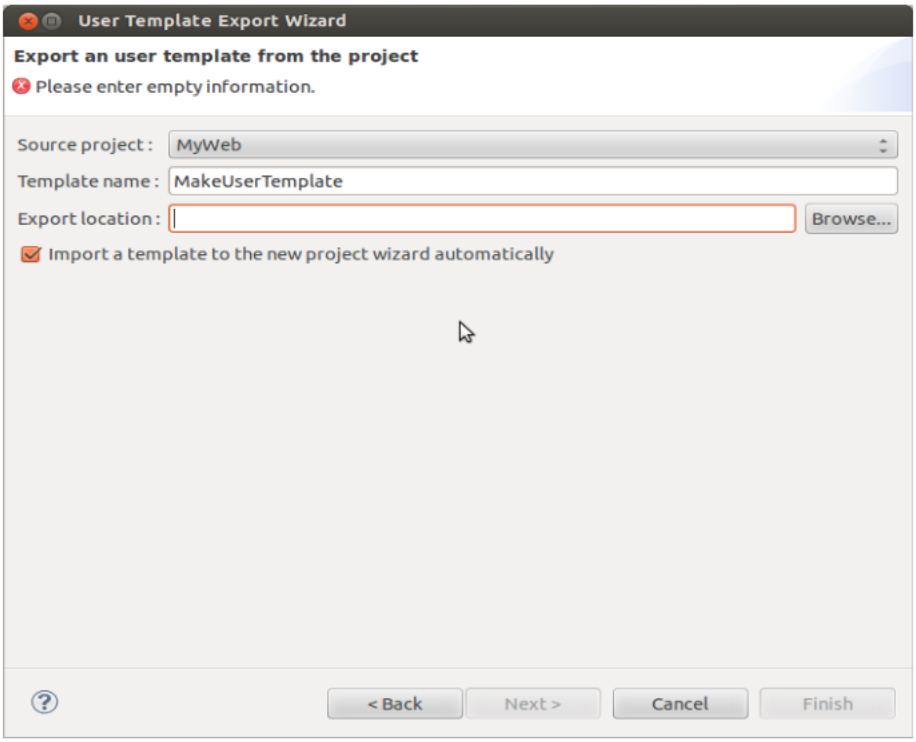

Figure 4: Defining the export location for a user template

d. To add the template to the **User Template** tab of the **New Tizen Web Project** window, select **Import a template to the new project wizard automatically**.

e. Optionally, to select a template description, preview image, and icon files to be shown in the Project Wizard, click **Next**.

3. Click **Finish**.

Setting Project Properties

Before implementing the actual application functionality, set all the necessary properties for your application project.

To set the project properties:

1. In the IDE, right-click the project in the **Project Explorer** view, and select **Properties**.
2. Set the following properties:
   - API privileges
   - Build properties
   - JSON properties
3. Click **Apply**.

> **Note:**
>
> Set the project properties only with the configuration editor in the IDE. If you create or edit the `config.xml` file using another editor, your application may not work as expected.

After you have finished setting the project properties, configure the application.

Checking API Privileges

To check the source code in your project for any violation of API privileges:

1. In the **Properties** window, select **Project > Properties > Tizen SDK > Web > Privilege**.
2. If you want to exclude JavaScript files in specific folders, define the folders in the **Excluded Folder Settings** panel.

   Excluding folders avoids unnecessary analysis of library files like `tizen-ui-fw` and `jQuery`. The `tizen-web-ui-fw` folder is excluded by default.
3. To perform privilege checks automatically, select the required option in the **Mode Settings** panel.
4. To perform privilege checks during the build process, select **Run privilege checks with build**.

You can also perform privilege checks manually by selecting **Project > Check Privilege** in the context menu.

The results are displayed in the **Problems** view.

Setting Build Properties

To set the build properties:

1. In the **Properties** window, select **Tizen SDK > Package > Web**.
2. Select **Optimize Web resources**.
3. In the **Optimization** panel, add any files to exclude from optimization.

Setting JSON Properties

To set the JSON properties:

1. In the **Properties** window, select **Tizen SDK > Web > JSON Properties**.
2. Select **Enable JSON validation in project**.

Configuring the Application

Every application includes the `config.xml` file, which contains various properties for configuring the application. To edit the `config.xml` file, open it by double-clicking the file in the **Project Explorer** view.

You can edit the following properties by using the corresponding tabs in the editor:

- Overview

  Define general information.
- Widget

  Define license information and UI preferences.
- Features

  Define software and hardware feature requirements.
- Privileges

  Define privileges for accessing restricted APIs or API groups.
- Policy

  Define network URLs for requesting external network resource permissions.

- Localization

  Define localization information for the name, description, and license elements of the `config.xml` file.

- Preferences

  Define key-value pairs that can be set and retrieved with the Widget API.

- Tizen

  Define Tizen-specific requirements information.

- Source

  View and edit the raw XML code of the `config.xml` file.

After you have finished configuring the application, create the application UI.

Overview

The **Overview** tab allows you to define general information for the application.

The following table lists the properties you can edit in the tab.

Table 3: Overview properties

Property	Description
Identifier	Application ID.
Version	Current version of the application.
Name	Application name that is displayed in an application menu or in other contexts.
Content	Startup file of the application.
Icon	Custom icon for the application, displayed in the main menu. The icon format must be 32-bit PNG with alpha, and the icon size must be 117 x 117 pixels.

Widget

The **Widget** tab allows you to define license information and UI preferences for the application.

The following table lists the properties you can edit in the tab.

Table 4: Widget properties

Property	Description
Managing the Widget	
Author	Person or organization that created the application.
E-mail	The author's email address.
Web Site	IRI associated with the application, such as a home page or a profile on a social network.
License	Software license under which the content of the application is provided. The license can include content, such as a usage agreement, redistribution statement, and copyright license terms.

License URL	Valid IRI or path associated with the software or content license.
Description	Free-form description of the application.
Managing the Widget UI	
Width	Startup file viewport width.
Height	Startup file viewport height.
View Modes	Preferred view mode.

Features

The **Features** tab allows you to define software and hardware feature requirements for the application. The definitions can be used for application filtering in the Tizen Store.

To add a required feature:

1. In the **Features** tab, click **Add**.
2. Select a feature from the list.
3. Click **Save**.

After adding the feature, you can see the code in the **Source** tab, for example:

```
<tizen:feature name="http://tizen.org/feature/network.nfc"/>
```

Privileges

The **Privileges** tab allows you to define privileges for accessing restricted APIs or API groups from the application. The tab functions as a standardized tool for requesting the binding of an IRI-identifiable runtime component for the application to use at runtime.

To add a privilege:

1. In the **Privileges** tab, click **Add**.
2. In the **Add privilege** window, set one of the following:
   - **Internal**: Select a privilege from the predefined list of APIs. You can select multiple privileges by pressing and holding the CTRL key.
   - **Privilege name**: Manually enter the URL containing a privilege definition.
   - **File**: Click **Browse**, and select a privilege file (`.xml` or `.widlprocxml`).
3. Click **Finish**.

After adding the privilege, you can see the code in the **Source** tab, for example:

```
<tizen:privilege name="http://tizen.org/privilege/application.launch"/>
```

Policy

The **Policy** tab allows you to define network URLs for requesting external network resource permissions.

According to the W3C Access Requests Policy (WARP), you cannot access external network

resources by default. If you need access to an external network resource, you must request a network resource permission for it.

The following table lists the properties you can edit in the tab.

Table 5: Policy properties

Property	Description

`content-security-policy`	Additional content security policy for a packaged or hosted application. The policy string is defined according to the W3C Content Security Policy 1.0.
	The following example shows the code in the **Source** tab:
	`<tizen:content-security-policy>`
	`  script-src 'self'`
	`</tizen:content-security-policy>`

`content-security-policy-report-only`	Additional content security policy for a packaged or hosted application for monitoring purposes only.
	The following example shows the code in the **Source** tab:
	`<tizen:content-security-policy-report-only>`
	`  script-src 'self';`
	`  report-uri="http://example.com/report.cgi"`
	`</tizen:content-security-policy-report-only>`

`allow-navigation`	List of URL domains allowed for the Web application.
	This property is optional.
	The following example shows the code in the **Source** tab:
	`<tizen:allow-navigation>`
	`  tizen.org *.tizen.org`
	`<tizen:allow-navigation/>`

`access`	Network resource permissions. To request a network resource permission: 1. Click **Add**. 2. In the **Network URL** column, enter the resource URL you want to access. 3. To allow the application to access the URL subdomains, set the value in the **Allow subdomain** column to **true** by clicking the value. The following example shows the code in the **Source** tab: `<access origin="http://www.tizen.org" subdomains="true"/>`

Localization

The **Localization** tab allows you to define localization information for the name, description, and license elements of the `config.xml` file.

To add localization information:

1. In the **Name** panel, click **Add**.
2. Select a locale, and enter your name.
3. In the **Description** panel, click **Add**.
4. Select a locale, and enter the description of the application.
5. In the **License** panel, click **Add**.
6. Select a locale, and enter the license and license URI.

After adding the localization information, you can see the code in the **Source** tab, for example:

```
<name xml:lang="en-gb">Lee</name>

<description xml:lang="en-gb">Widget</description>

<license xml:lang="en-gb" href=" http://www.apache.org/licenses/LICENSE-
2.0.html">
   Apache License, Version 2.0
</license>
```

Preferences

The **Preferences** tab allows you to define key-value pairs that can be set and retrieved with the Widget API. The application uses these key-value pairs during runtime.

To add a key-value pair:

1. In the **Preferences** tab, click **Add**.
2. Enter the key in the **Name** column and its value in the **Value** column.
3. To set the key-value pair as read-only, set the value in the **Read-only** column to **true** by clicking the value.

After adding the key-value pair, you can see the code in the **Source** tab, for example:

```
<preference name="key" value="value" readonly="false"/>
```

Tizen

The **Tizen** tab allows you to define Tizen-specific requirements information. The tab shows the Tizen schema extension. Some of the properties on this tab are mandatory, others are optional.

The following table lists the properties you can edit in the tab.

Table 6: Tizen properties

Property	Description
application	
ID	Tizen application ID, which is randomly generated from the Tizen package ID and project name. This property is mandatory.
Required version	Indicates the minimum API version that the application supports. This property is mandatory and must be a float value, such as 1.0 or 2.0.
content	
src	In the W3C Widget Packaging and XML Configuration, the Web application start page is a document within the widget package. The Tizen WRT allows the start page to be hosted on an external server. The `<tizen:content />` element is used to point to the relevant document.
setting	
screen-orientation	Screen orientation mode: • **landscape**

	• **portrait** • **auto-rotation** This property is optional. The default value is **portrait**.
context-menu	Support for context menus. This property is optional. The default value is **enable**.
background-support	Sets whether the execution of the application continues when the application is sent to the background. This property is optional. The default value is **disable**.
encryption	Encryption of application resources (HTML, CSS, and JavaScript files). This property is optional. The default value is **disable**.
install-location	Installation location, such as an SD card. This property is optional. The default value is **auto**.
hwkey-event	Support for hardware keys. This property is optional. The default value is **enable**.
app-control	

	Describes the application control functionalities provided by the application. The **operation**, **uri**, and **mime** fields describe the functionalities that other applications can request, and the **src** field describes the application page that handles the request.
	The following example shows the code in the **Source** tab:
	`<tizen:app-control>`
	`  <tizen:src name="edit.html"/>`
	`  <tizen:operation name="http://tizen.org/appcontrol/operation/edit"/>`
	`  <tizen:uri name="file"/>`
	`  <tizen:mime name="image/jpeg"/>`
	`</tizen:app-control>`

app-widget	
id	Dynamic Box ID. The ID format is **[<tizen:application> id].[any string]**.
primary	When you click the application icon to add a Dynamic Box, and use the **Add from app tray** option, the primary Dynamic Box is displayed on the home screen. You can add a secondary Dynamic Box by using the **Add more app-widget** option. Only one Dynamic Box can have the value of this property set to **true**.
auto-launch	To launch the application when the Dynamic Box is clicked, set the value of this property to **true**.
update-period	Used to determine how often (in seconds) the Dynamic Box is reloaded. The minimum value is 1800.0. The default behavior is **no-update**.
box-label	Name of the Dynamic Box.
box-icon	This icon is displayed on the list when you select the **Add more app-widget** option.
box-content	The **src** attribute specifies the relative path of the start page based on the application root folder. The **mouse-event** attribute specifies the flag that indicates that the Dynamic Box content receives mouse down and up events.

	The **touch-effect** attribute specifies the flag that indicates that a special UI effect must be applied to the Dynamic Box when the user clicks the Dynamic Box.
box-size	The **preview** attribute specifies the path of the image file displayed on the Dynamic Box. The **use-decoration** attribute specifies whether the Dynamic Box Viewer applies a frame decoration to the box size.
pd	The **src** attribute specifies the relative path of the start page based on the application root folder, and is used to display the Drop View. You can also define the Drop View height. The Drop View width is always fixed to the device screen width.
account	
Display name	Display name of the account provider. This property is mandatory.
Multiple account	Sets whether multiple accounts are supported. This property is mandatory.
Icon	Path of the icon representing the account provider. The icon image is used by account settings and must be placed in a shared folder. The size is 72 x 72 pixels. This property is mandatory.
Small icon	Path of the small icon representing the account provider. The icon image is used by account settings and must be placed in a shared folder. The size is 45 x 45 pixels. This property is mandatory.
Capabilities	Capabilities of the account provider defined in the IRI format: `http://<vendor information>/accounts/capability/<name>` The following predefined capabilities can be used: • `http://tizen.org/account/capability/contact` Use when the account is related to contacts. • `http://tizen.org/account/capability/calendar` Use when the account is related to calendar. This property is optional.
metadata	
key	Metadata that can be accessed (read-only) through the Tizen Application API. The following example shows the code in the **Source** tab: `<tizen:metadata key="key1"/>` This property is optional. The value must be a unique string.
value	Metadata that can be accessed (read-only) through the Tizen Application API. The following example shows the code in the **Source** tab: `<tizen:metadata key="key2" value="value"/>` This property is optional. The value must be a string.

Source

The **Source** tab allows you to view and edit the raw XML code of the `config.xml` file. Any changes you make on the other tabs are reflected in the `config.xml` file.

Note:

The `config.xml` file must conform not only to the XML file format but also to the W3C specification requirements. Editing the configuration file in XML format is intended for advanced users only.

Creating the Application UI with the UI Builder

The Tizen IDE provides the UI Builder for creating application UIs. The UI Builder simplifies the UI creation process by allowing you to arrange widgets using a drag-and-drop WYSIWYG (What You See Is What You Get) editor.

For more information about creating the application UI using other methods, see Using the Tizen Advanced UI.

To create your application UI with the UI Builder:

1. Create a UI Builder project
2. Add a new page
3. Design the page by adding and arranging UI widgets
4. Handle events for the UI widgets
5. Test the UI Builder project

For more information about the UI Builder, see the UI Builder in Tizen Dev Guide.

Creating a UI Builder Project

A UI Builder project creates the skeleton source code and required files and folders in the IDE workspace for building a Tizen Web application. You can use the UI Builder to create and modify the page layout. The UI Builder project uses the Tizen Web UI Builder framework as a guide for the programming model.

Creating a UI Builder project is similar to creating a Web application project. In the **New Web Project** window, select a Tizen Web UI Builder project template.

Adding a Page

You can add multiple pages to a UI Builder application.

To add a new page:

1. In the IDE, select **File > New > Other**.
2. In the **New** window, select **Tizen > Tizen Web UI Builder Page**.
3. In the **New Page File** window, select the UI Builder project for the page and enter the new page ID.
4. Click **Finish**.

The new page is shown in the **Page Designer** view.

Designing the Page

You use the **Page Designer** view to add and remove UI widgets to and from the page, arrange the widgets to the layout you want, and set the widget properties. You can add many different types of

widgets to the page.

You can configure the design area for the page by setting the target device resolution, zooming in and out, and scaling the design area to the available screen space.

To add a widget to the page:

1. In the IDE, in the **Pages** view, click the thumbnail image of the page to which you want to add the widget.

   The **Page Designer** view displays the design area for the page.

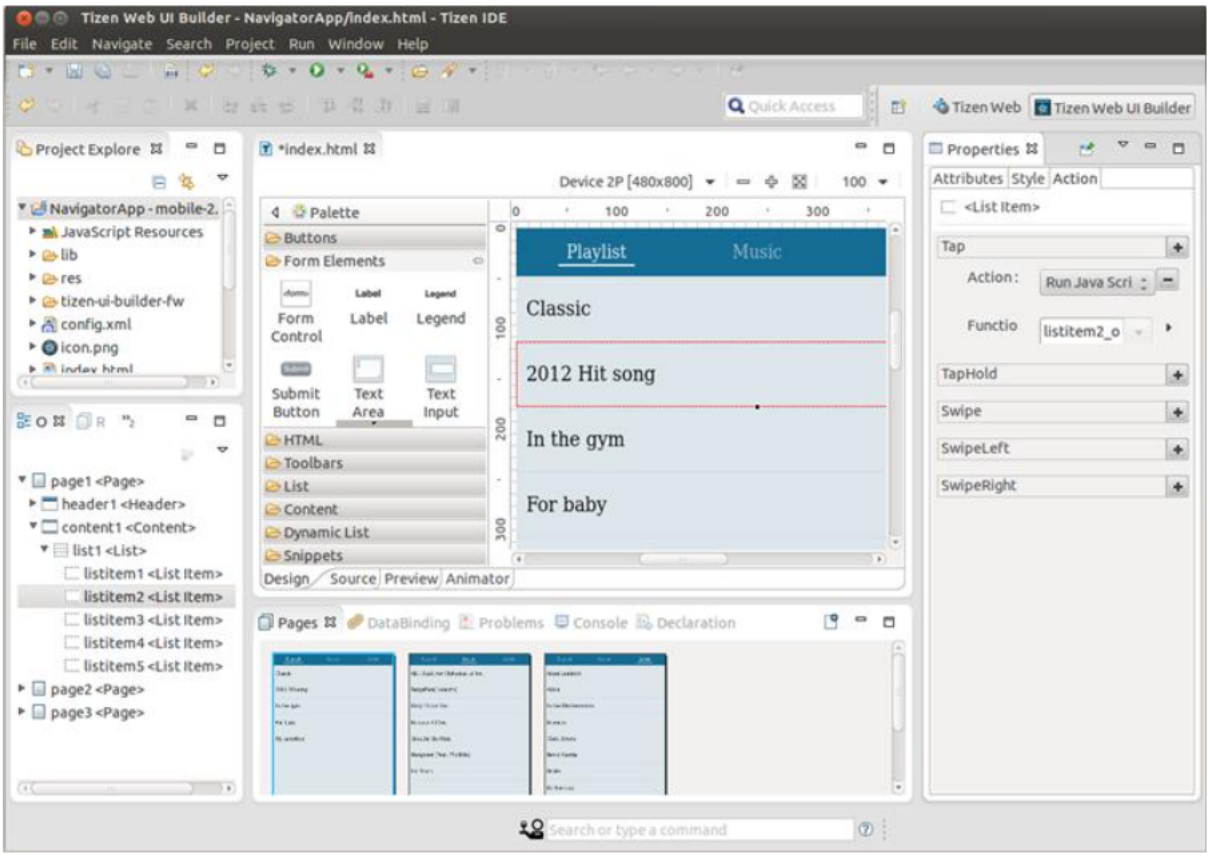

Figure 5: UI Builder project in the Tizen IDE

2. Drag the widget from the **Palette** area to the design area.

   The next empty position in the design area is marked with a dotted rectangle. The parent widget or content object is colored blue. For more information about adding widgets, see Placing Widgets.

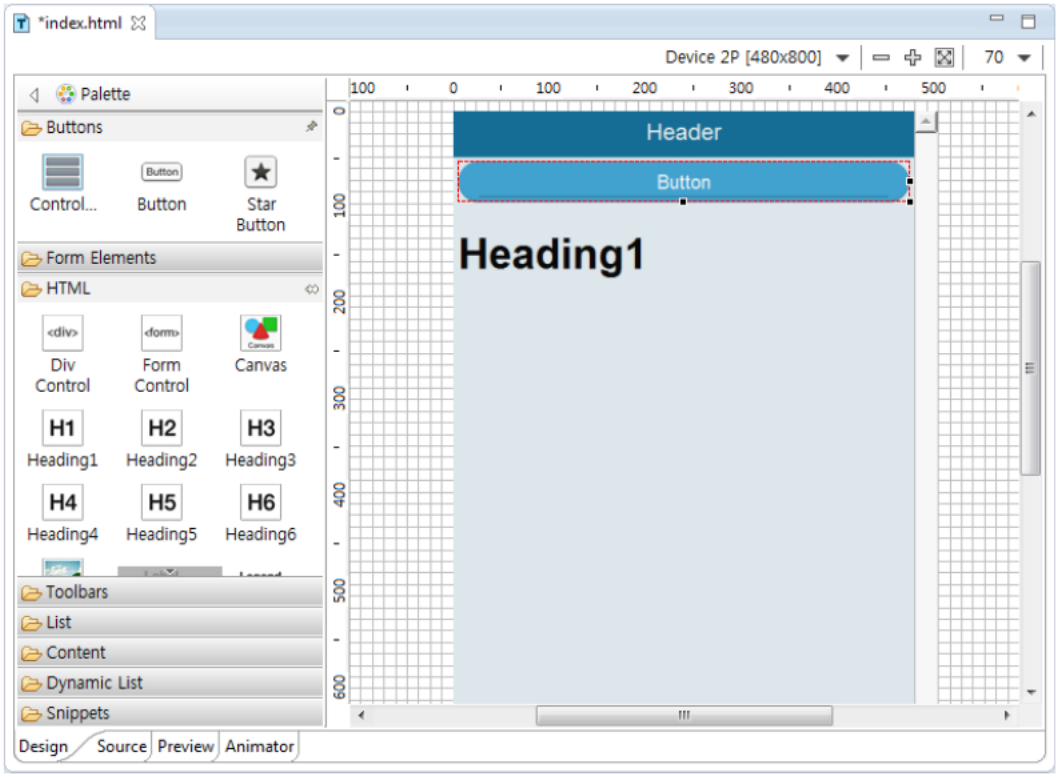

Figure 6: Adding a widget from the Palette to the page

3. Select the widget in the design area or in the **Outline** view.

   The following figure show a close-up of the **Outline** view, with list item widgets nested inside a list widget.

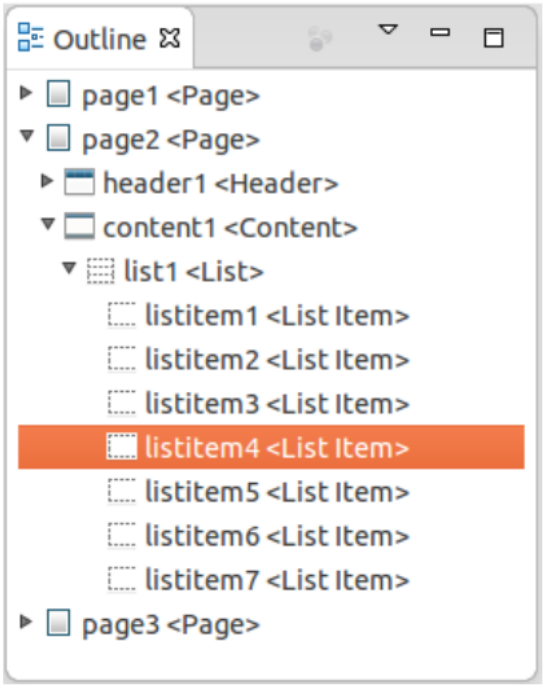

Figure 7: Outline view

4. In the **Properties** view, set the widget properties.

   The following figure shows the **Properties** view for a list item widget, with the list item text highlighted for editing.

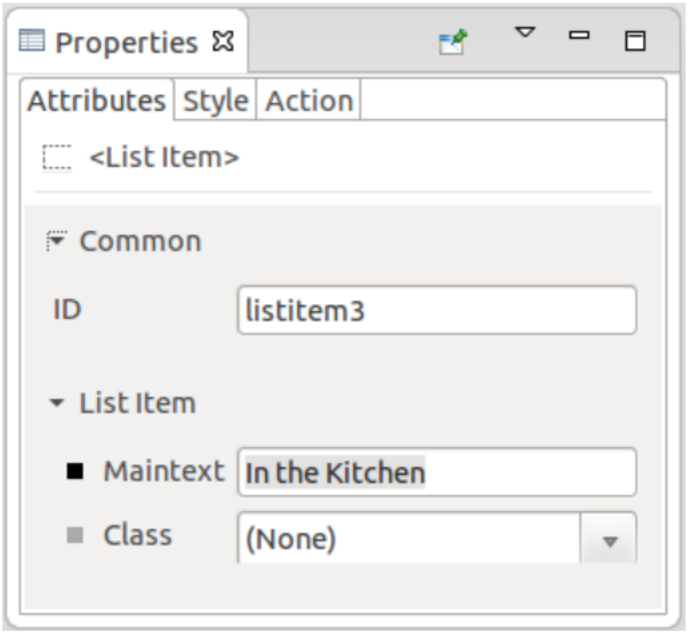

Figure 8: Properties view

5. Move the widget in the design area as necessary.

Placing Widgets

You can add child widgets to container widgets, such as a **List**. For example, you can add a set of **ListItem** widgets to a **List** widget. If a **List** widget placed in the design area does not contain any **ListItem** widgets, the **List** widget displays the message "Drop a List Item".

You can only add widgets of a specific type to a container widget. For example, you can add a **ListItem** widget only to a **List** widget. If you try to add a **ListItem** widget to a different type of widget, the design area changes to red, indicating that the widget cannot be placed there. If you add a widget to the correct type of container widget, the design area changes to blue, indicating a valid placement.

Handling Events on the Page

To add an event handler for a widget:

1. In the IDE, in the design area of the **Page Designer** view, select the widget.
2. In the **Properties** view, select the **Action** tab.
3. Click the plus button of the event for which you want to add the event handler, and select **Run JavaScript Function** from the **Action** drop-down list.
4. In the **Function** field, define a name for the event handler, and click the play button next to the field.

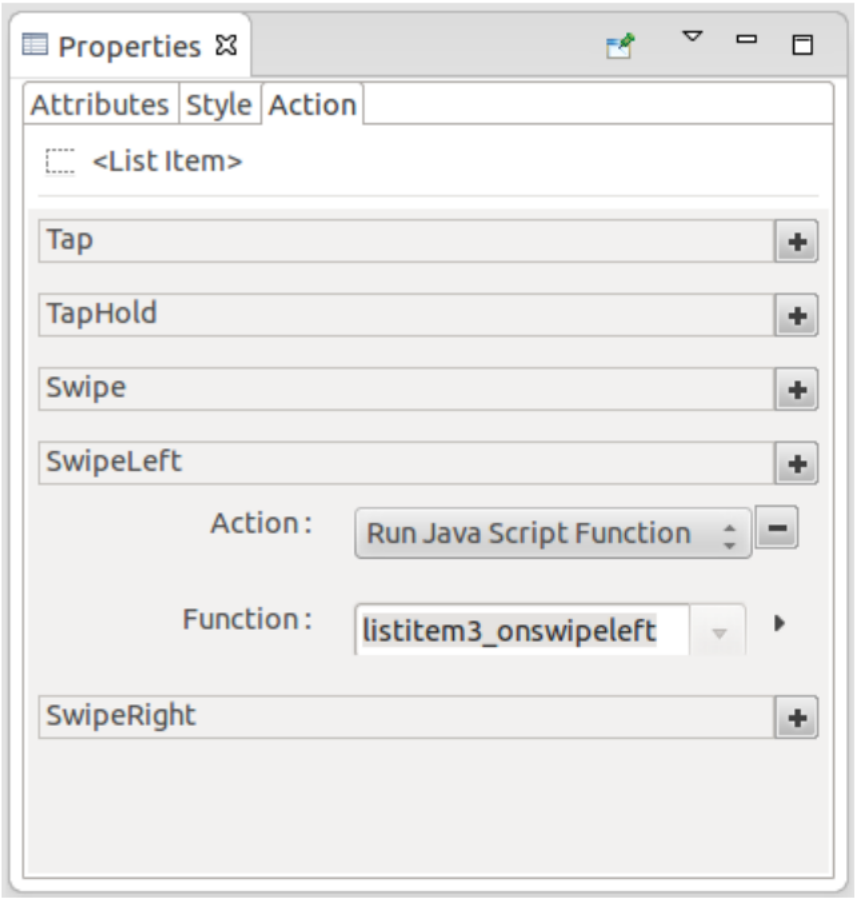

Figure 9: Adding the SwipeLeft handler to the list item.

The UI Builder generates a skeleton code for the event handler, for example:

```
/**
* @param {Object} event
* @base _page1_page
* @returns {Boolean}
*/
_page1_page.prototype.listitem3_onswipeleft = function(event) {
};
```

5. Enter the application-specific event handler code.

   Each widget has its own unique ID, and you can access the widget from the event handler code by using that ID. You can view and edit the widget ID in the **Outline** view.

   For example, the following code displays an alert window when the user swipes left over the `listitem3` widget:

```
_page1_page.prototype.listitem3_onswipeleft = function(event)
{
   window.alert("Hello World");
};
```

Testing the UI Builder Project

To test the UI Builder project:

1. In the IDE, in the **Project Explorer** view, select the project.
2. To test the project in the Emulator or on a target device, select **Run > Run as > Tizen Web Application**.
3. To test the project in the Simulator, select **Run > Run as > Tizen Web Simulator Application**.

Building the Application

Before you can run and debug the application, you must first build it. You can build the application automatically or manually:

- To build the application automatically, in the IDE, select **Project > Build Automatically**.

  If you select this option, whenever the source code or a resource is changed and saved, the IDE automatically rebuilds the application.

- To build the application manually, in the IDE, select **Project > Build Project**.

  If you select this option, you can build your project at your convenience.

Note:

If you select to build the application manually:

- Make sure you have the latest build output before you run or debug the project.
- To remove a project build output, in the IDE, select **Project > Clean**.

Running and Debugging the Application

After building your application, run and debug it in one or more of the following environments:

- **Simulator**

  The Tizen Web Simulator allows you to run applications that use the Tizen Web APIs.

- **Emulator**

  The Tizen SDK Emulator imitates the target environment for running Tizen Web applications. Using this environment, you can test your application before deploying it to an actual target device.

- **Target device**

  Running your application on a target device allows you to debug and test your application in the actual target environment.

You can make running and debugging your application faster by using Rapid Development Support (RDS). For more information about the debugging methods and tools you can use, see Debugging the Application.

After you have finished debugging your application, you are ready to package it.

Running the Application in the Simulator

Note:

The Tizen Web Simulator requires Google Chrome to run. To use the Simulator, first download and

install Google Chrome, and then specify the browser installation location in the simulator preferences.

To run your application in the Simulator:

1. If you are running the application for the first time, create a run configuration by selecting **Run > Run Configurations > Tizen Web Simulator Application** from the menu. The run configuration contains the necessary application launch settings.
2. Run the application by doing one of the following:
   - In the **Project Explorer** view, right-click the project, and select **Run As > Tizen Web Simulator Application**.
   - In the menu, select **Run > Run As > Tizen Web Simulator Application**.
   - In the toolbar, click **Run**.

When the application is launched, the Simulator loads the file specified in the **Content** field of the `config.xml` file. The most commonly specified file is `index.html`.

The Simulator renders your application in the browser using WebKit. All Google Chrome development features are available in the Simulator, including the Remote Inspector tool, which you can open by pressing the **F12** key.

The Simulator allows you to set the device screen size and orientation, and send events and messages, such as geolocation data and sensor input events, to your application for debugging purposes.

Running the Application in the Emulator

Note:

Running the Emulator requires a minimum free disk space of 20 MB. The image can take up to 10 GB of disk space.

To run your application in the Emulator:

1. Launch the Emulator:
   a. Start the Emulator Manager from the Desktop (Linux) or Start Menu (Windows), or from the command line. The Mac OS® X does not provide a shortcut for the Emulator Manager.
   b. In the Emulator Manager, select the Emulator image in the **Name** column.

      If you are using the Emulator Manager for the first time, create an Emulator image:

      i. Click **Create New**.
      ii. Set the image details.
      iii. Click **Save**.

   c. Click **Launch Emulator**.
2. Run the application by doing one of the following:
   - In the **Project Explorer** view, right-click the project, and select **Run As > Tizen Web Application**.
   - In the menu, select **Run > Run As > Tizen Web Application**.
   - In the toolbar, click **Run**.
3. To stop the Emulator, right-click the Emulator, and click **Close**.

Running the Application on a Target Device

To run your application on a target device:

1. Connect the target device to your computer.
2. In the IDE, select **Run > Run Configurations**.
3. In the **Run Configurations** window, click **New Launch Configuration**.

   Rapid Development Support is used to skip the package upload and make running the application faster. RDS is enabled by default. To disable it, select **Window > Preferences > Tizen SDK > Rapid Development Support**.
4. Set the timeout using the **Timeout** value slider.

   The timeout value represents the waiting time for the application launch. If you are using a lower configuration computer, set a higher timeout value to avoid application launch failure errors.
5. Click **Run**.

If no changes are required in the run configuration, you can run the application on the target device by doing one of the following:

- In the **Project Explorer** view, right-click the project, and select **Run As > Tizen Web Application**.
- In the menu, select **Run > Run As > Tizen Web Application**.
- In the toolbar, click **Run**.

If your application successfully launches on the target device, the **JavaScript Log Console** view is automatically launched in the IDE. The view displays the application JavaScript logs.

Debugging the Application

Debugging your application allows you to understand its control flow. You can debug the application by running it on a target device and debugging its JavaScript code. JavaScript code debugging uses the Remote Inspector tool.

To start debugging your application on the target device:

1. Connect the target device to your computer.
2. In the IDE, open the **Debug Configurations** window by doing one of the following:
   - In the **Project Explorer** view, right-click the project, and select **Debug As > Debug Configurations**.
   - In the menu, select **Run > Debug Configurations**.
3. In the **Debug Configurations** window, click **New Launch Configuration**.

   Rapid Development Support (RDS) is used to skip the package upload and make running the application faster. RDS is enabled by default. To disable it, select **Window > Preferences > Tizen SDK > Rapid Development Support**.
4. Set the timeout using the **Timeout** value slider.

   The timeout value represents the waiting time for the application launch. If you are using a lower configuration computer, set a higher timeout value to avoid application launch failure errors.
5. Click **Debug**.

If no changes are required in the debug configuration, you can debug the application on the target device by doing one of the following:

- In the **Project Explorer** view, right-click the project, and select **Debug As > Tizen Web Application**.
- In the menu, select **Run > Debug As > Tizen Web Application**.
- In the toolbar, click **Debug**.

Once the debugging starts, the Remote Inspector is displayed in a new Google Chrome window. You can perform the following debugging tasks using the Remote Inspector:

- Inspect styles
- Inspect the DOM
- Inspect resources
- Debug JavaScript code

Note:

The Remote Inspector always opens in a new browser window. Life-cycle synchronization between the application to be debugged and the Remote Inspector is not supported.

Installing Google Chrome on the device is mandatory for the Remote Inspector to work. When Google Chrome is installed on the device, the Tizen SDK automatically detects it. To select the browser path, go to **Window > Preferences > Tizen SDK > Web > Chrome**.

Debugging JavaScript Code

Note:

You must enable debugging before debugging JavaScript code.

To debug the JavaScript code, click **Sources** from the Remote Inspector menu.

You can set a break point in the code by right-clicking in the marker bar area on the left side of the editor, and selecting **Toggle Breakpoint**.

Once the break points are set, you can watch variables, expressions, and the current call stack. You can also control the debugging by using the following control buttons.

Table 7: Control buttons for debugging between break points

Button	Description
	Resumes the current execution.
	Steps over the highlighted statement. Executes the current line, and if the line contains a method, executes the method without entering it.
	Steps into the highlighted statement. Executes the current line, and if the line contains a method, steps into the method.
	Steps out of the current method.

	Deactivates all break points.

If your application successfully launches on the target device, the **JavaScript Log Console** view is automatically launched in the IDE. The view displays the application JavaScript logs.

Rapid Development Support

Rapid Development Support (RDS) allows you to develop a Tizen application rapidly by saving deployment time.

When using RDS, the application is first launched normally (normal mode). After the first launch, the packaging process is skipped and only modified, added, and deleted files are installed on the target device (RDS mode). If an error occurs at launch time, the application is launched in normal mode.

To launch the application in normal mode:

1. Package the application.
2. Transfer the packaged file to the target device.
3. Install the packaged file on the target device. If the application is already installed, uninstall it before the installation.

To launch the application in RDS mode:

1. Search for the delta files (changed, added, and deleted files).
2. Transfer the delta files to the target device.
3. If the `config.xml` file has been modified, run folder installation.

The RDS option is enabled by default. To disable it, in the IDE, select **Window > Preferences > Tizen SDK > Rapid Development Support**.

Note:

RDS is not supported in multi-application projects.

If you want to remove an application from your device, you must manually delete the installed application, since the launch process does not include uninstallation.

Packaging the Application

After successfully running and debugging your application, you need to package it. You can package your application and enable a single-download-and-installation. The application package manager provided by the IDE allows you to install and uninstall the package in the Emulator or on a target device.

Note:

The application must be signed with an author certificate. If you do not have a certificate, create it using the certificate generator, and register it in the IDE.

Web application packaging is based on the W3C packaging and configuration specification. You can manage packaging using the following build options:

- **Automatic**

  When using the automatic build option, the IDE automatically recognizes any saved changes in the source or resources, and rebuilds the project. The automatic build option is enabled by default. To disable or enable the option, in the IDE, select **Project > Build Automatically**.

- **Manual**

  The manual build option allows you to build your project manually using your changes. To use this option:

  a. In the IDE, disable the automatic build option by selecting **Project > Build Automatically**.

  b. Remove the project build output by selecting **Project > Clean**.

  c. Rebuild the project by selecting **Project > Build Project**.

> **Note:**
>
> When using the manual build option, make sure the build output is the latest version before running or debugging the project.

You can also package your application by running the `web-packaging` command from the command line:

```
web-packaging project.wgt project/
```

By default, the application package is created once. You can view the package content at any point in the application development process.

Viewing the Application Package

To view the contents of an application package, double-click the project `.wgt` file in the **Project Explorer** view. All the files present in the application project are displayed in a list.

Any changes made to the files in the package content list, such as deleting files or dragging and dropping files, are not reflected in the actual project files.

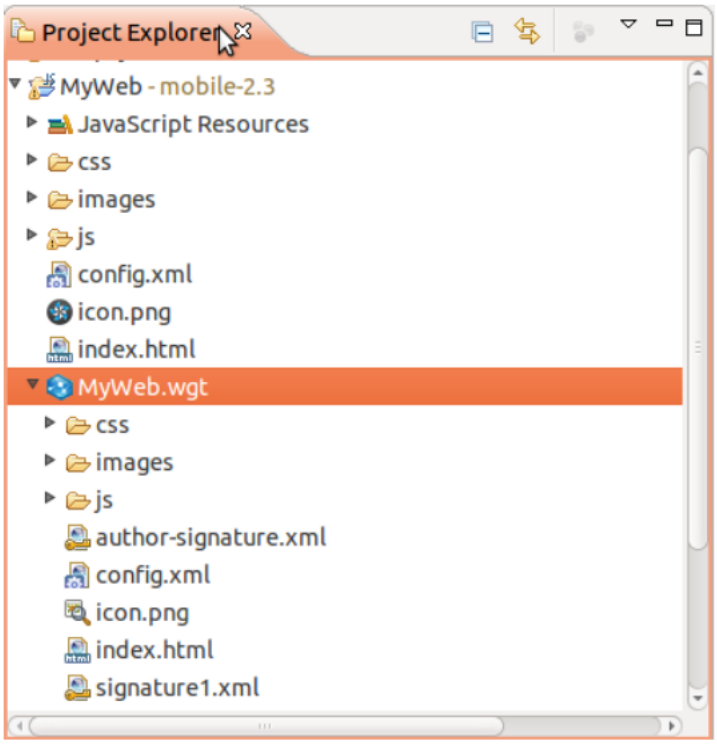

Figure 10: Viewing the contents of an application package

Localizing the Application

You can localize your application for different languages and cultures by creating separate application versions for different locales.

To localize your application:

1. Launch the Localization Wizard by doing one of the following:
   - In the **Project Explorer** view, right-click the project, and select **Localization > Localization Wizard**.
   - In the menu, select **Project > Localization > Localization Wizard**.
2. In the Localization Wizard, select the files to localize from the list of files, and click **Next**.

   The files that are already localized are grayed out.
3. In the **Available locales** list, select the locales and add them to the **Selected locales** list. Click **Next**.

   The locales that are already supported are grayed out.
4. Associate files with specific locales by selecting the check boxes under the desired locale column.

   The check boxes for files that are already localized for a particular locale are grayed out.
5. Click **Finish**.

   In the **Project Explorer** view, a new `locales` folder is created containing separate subfolders for each locale. The subfolders contain copies of the selected files. Use these copies to create localized versions of the files.

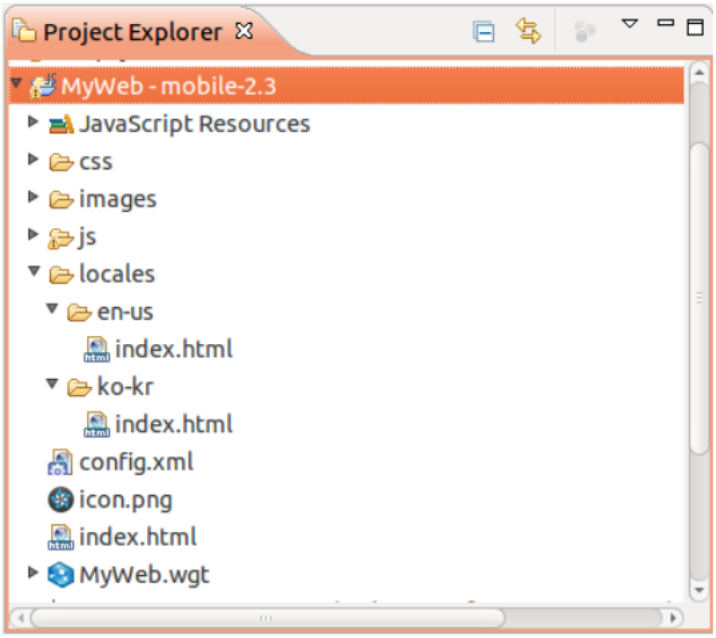

Figure 11: Locale subfolders

5. Using the Tizen Advanced UI

This chapter introduces the Tizen Advanced UI (TAU) framework and the basic UI elements it provides, and shows you how Tizen's support for the latest CSS features allows you to create flipboard-like effects.

For information about creating the application UI using the UI Builder, see Creating the Application UI with the UI Builder.

Introduction to the Tizen Advanced UI

Tizen Advanced UI is an HTML, CSS, and JavaScript framework for developing Web applications. The framework provides UI elements that give your applications a unique Tizen flavor. The framework also provides event and utility methods for handling the UI elements.

Because HTML already supports meaningful primitive elements, such as header, content, footer, and button, TAU only provides themes for the primitive elements as well as some additional features.

Getting Started with a Simple Application

To get started with a simple single-page Web application:

1. Create a Web application project and select **Tizen Web UI Framework > Single Page Application** as the application template.

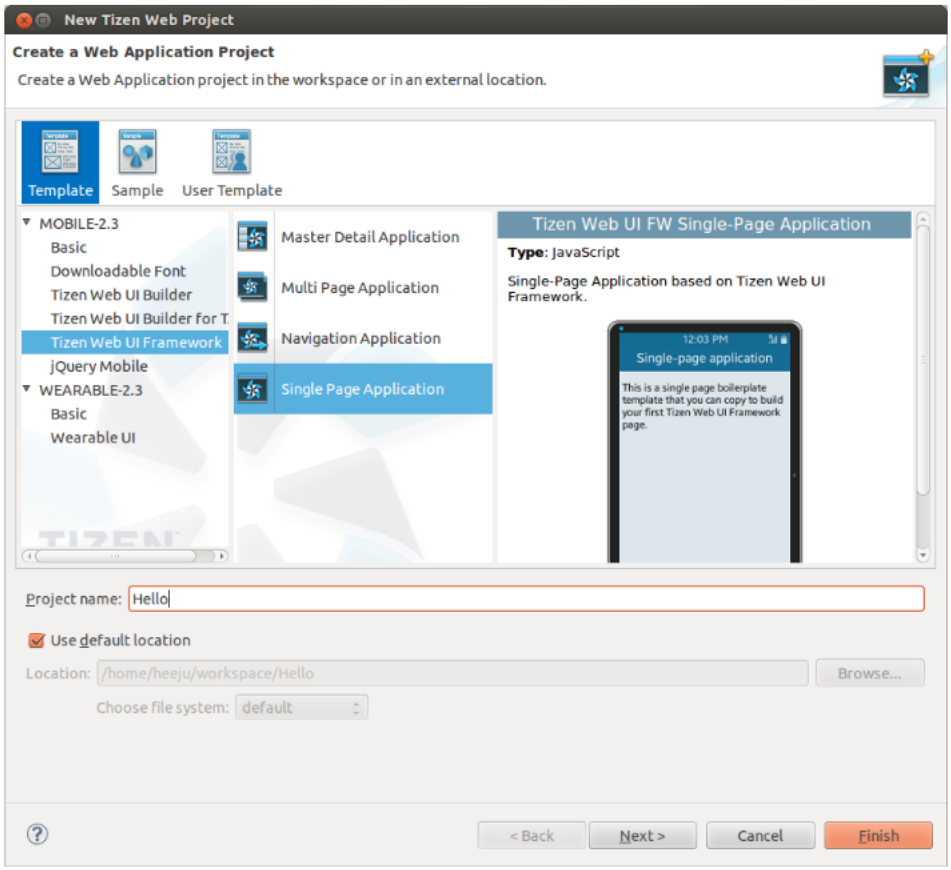

Figure 12: Selecting a template for a new Web application project

2. Declare the viewport meta tag for the application:

```
<meta name="viewport" content="..." />
```

The viewport is the screen area in which the Web engine displays the UI. The viewport meta tag informs the Web engine that the application is targeted for a specific form factor, such as a specific screen width. This allows the Web engine to determine the correct scaling factor for the application UI on the current device and screen. For example, to specify a target width of 360 pixels for your application, declare the following viewport meta tag:

```
<meta name="viewport" content="width=360" />
```

If you are creating a responsive application that dynamically adapts its UI for each device, declare the following viewport meta tag:

```
<meta name="viewport" content="width=device-width" />
```

The `device-width` value sets the target width to the ideal width for the current device's screen.

In addition to `width`, you can also specify other form factor attributes, such as `height`, `initial-scale`, `minimum-scale`, `maximum-scale`, and `user-scalable`, in the viewport meta tag. However, use these other attributes with caution. Wrong values can cause the Web engine to display the UI incorrectly.

3. Import the TAU CSS and JavaScript files:

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width" />
<link rel="stylesheet" href="lib/tau/mobile/theme/default/tau.min.css">
<script type="text/javascript" scr="lib/tau/mobile/js/tau.min.js">
...
</head>
<body>
...
</body>
</html>
```

The template provides a basic skeleton for your application code, so you can now start developing the application.

Adding a Page

To display the UI on the device screen, you need to add at least one view to the application. Each view takes up the full screen. In TAU, a view is declared as a page.

To add a page to the application, use the following code:

```
<div data-role="page"><!--Page area (mandatory)-->
   <div data-role="header"><!--Header area (optional)--></div>
   <div data-role="content"><!--Content area (optional)--></div>
   <div data-role="footer"><!--Footer area(optional>--></div>
</div>
```

The page is declared as a `<div>` element with the `data-role="page"` attribute. Inside the `<div>` element, you can define the header, content, and footer for the page, as shown in the following figure.

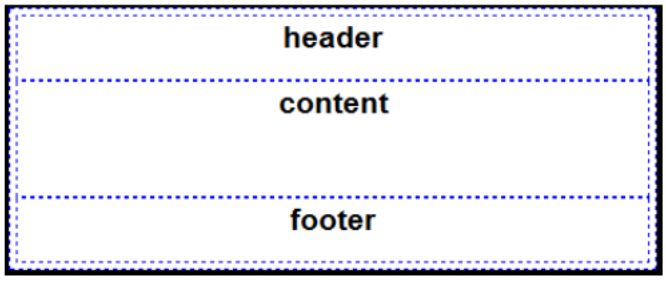

Figure 13: Page structure

Handling Events on the Page

TAU also provides various custom events for creating hooks between the application UI and the JavaScript code. These events complement the native events of the platform and include, for example, touch-based gesture events, such as swipe, drag, and pinch.

For example, to handle swipe and drag events in your UI, use the following code as a basis:

```
var gesture = new Gesture(element,
{
   dragOrientation: tau.gesture.Orientation.VERTICAL
})
.on("drag dragend dragcancel swipe", function(event)
{
   /* Handle swipe and drag events */
});
```

You can also use jQuery with TAU.

Available UI Widgets

The following table lists the UI widgets available in TAU.

Table 8: UI widgets available in TAU

Widget	Description
Autodividers	The autodividers widget automatically creates dividers for a list view widget as an extension.
Button	The button widget changes the default browser buttons to special buttons with additional features, such as icons, corners, and shadows.
Checkbox radio	The checkbox radio widget changes the default browser checkboxes and radio buttons to a form more adapted to the mobile environment.
Collapsible	The collapsible widget allows you to expand or collapse content when tapped.
Control group	The control group widget improves the styling of multiple buttons by grouping them into a single block.
Date-time picker	The date-time picker widget shows a control that the user can use to enter date and time values.
Drawer	The drawer widget allows you to open and close a drawer to show or hide the content inside it.

Fast scroll	The fast scroll widget shows a shortcut list that is bound to its parent scroll bar and respective list view.
Flip toggle switch	The flip toggle switch widget is a common UI element used for binary on/off or true/false data input.
Gallery	The gallery widget shows images in a gallery on the screen.
List divider	The list divider widget creates a list separator, which can be used for grouping list items.
List view	The list view widget is used to display, for example, navigation data, results, and data entries, in a list format.
Loader	The loader widget displays a loader pop-up on page changes.
Notification	The notification widget shows a pop-up on the screen to provide notifications.
Popup	The popup widget supports two pop-ups: the position-to-window pop-up (like a system pop-up) and the context pop-up.
Progress	The progress widget shows that an operation is in progress.
Progress bar	The progress bar widget shows a control that indicates the progress percentage of an on-going operation.
Scroll handler	The scroll handler widget is an extension for the scroll view widget, and adds a scroll button that looks like a handle.
Search bar	The search bar widget is used to search for page content.
Slider	The slider widget changes the range-type browser input to sliders.
Swipe	The swipe widget shows a list view on the screen where the list items can be swiped vertically to show a menu.
Tab bar	The tab bar widget shows an unordered list of buttons on the screen wrapped together in a single group.
Token text area	The token text area widget changes a text item to a button.
Virtual list view	The virtual list view widget is used to display data elements in a list format with dynamic data management.

Handling Multiple Pages, Widgets, and Events

The following examples show you how to create an application with two pages, one button widget on each page, and event handlers for changing the currently displayed page:

1. Add a page to the application by using the UI Builder or manually editing the HTML code:

```
<div data-role="page" id="page1">
   <div data-role="header">
      <h1>Page 1</h1>
   </div>
   <div data-role="content">
```

```
      Swipe right or
      <button id="button1" data-role="button">Press button</button>
      to change to page 2
   </div>
</div>
```

The above code adds a page with some text and a button widget.

2. Add a second page to the application, also with some text and a button widget:

```
<div data-role="page" id="page2">
   <div data-role="header">
      <h1>Page 2</h1>
   </div>
   <div data-role="content">
      Swipe left or
      <button id="button2" data-role="button">Press button</button>
      to change back to page 1
   </div>
</div>
```

3. Define the event and event handlers for changing the currently displayed page when the user either swipes left or right across the screen or clicks a button:

```
<script>
tau.event.one(document, "pageshow", function()
{
   /* Swipe handlers */
   $("#page1").on("swipeleft", function(event)
   {
      tau.changePage("#page2");
   });
   $("#page2").on("swiperight", function(event)
   {
      tau.changePage("#page1");
   });

   /* Button click handlers */
   $("#button1").on("click", function(event)
   {
      tau.changePage("#page2");
   });
   $("#button2").on("click", function(event)
   {
      tau.changePage("#page1");
   });
})
</script>
```

The `tau.changePage()` method changes the currently displayed page to the page with the specified ID. If the user swipes left or right or clicks the button on page 1, the application switches to page 2, and vice versa.

6. Using Tizen Web APIs

This chapter shows you how to use Tizen Web APIs to develop the following types of features:

- Calendar
- Contact
- Messaging
- Multimedia
- NFC

Calendar

The Calendar API allows you to manage events and tasks in a calendar.

A calendar is a collection of events or tasks, depending on the calendar type. Each event and task has a series of attributes, such as purpose, starting time, and duration.

The events and tasks are identified using the `CalendarItemId` type definition, which is either a `CalendarTaskId` (for tasks) or `CalendarEventId` (for events). In recurring events, the `CalendarEventId` contains a recurrence ID (RID) in addition to the actual event ID for separately identifying each occurrence of the event.

The Calendar API uses the `TZDate` object of the Time API, rather than the standard JavaScript `Date` object, to handle difficult issues related to the time zone, because the `TZDate` object handles exact time and provides various utility methods.

> **Note:**
>
> Due to time zones and daylight saving time, an event for "today" can actually occur in the past or in the future.

Calendar API Main Features

The main features of the Calendar API include:

- **Calendar management**

  To access a calendar item, you must first retrieve the `Calendar` object of the applicable type. To access the device calendars, you can use:

  - The `getDefaultCalendar()` method of the `CalendarManager` interface to retrieve the default calendar
  - The `getCalendars()` method to retrieve all the available calendars as an array
  - The `getUnifiedCalendar()` method of the `CalendarManager` interface to retrieve the special calendar which combines events and tasks from all calendars of the same type

- **Calendar item management**

  You can manage calendar items (add a new item to a calendar or manage a single calendar event) using the applicable methods of the `Calendar` interface.

To update or delete a single instance of a recurring event, first get the list of event instances with the `expandRecurrence()` method of the `CalendarEvent:CalendarItem` object. Then, delete the applicable event instance, or update it by calling the `update()` method with the `updateAllInstances` parameter set to `false`.

You can create and manage multiple calendar items simultaneously using the applicable batch methods: `addBatch()`, `updateBatch()`, and `removeBatch()`. The batch mode provides faster, optimized processing of multiple calendar items.

When searching for calendar items, you can create attribute filters, attribute range filters, and composite filters based on specific filter attributes. You can also sort the search results.

Note:

The batch mode does not provide progress information about operations. To ensure that you can view the progress, break the batch operation down into multiple smaller batch operations. For example, break down a batch of 100 update requests into 10 batch operations that update 10 records at a time. Breaking down a batch operation also helps you avoid blocking other database operations, such as add or remove.

- **Calendar item alarms**

  You can set an alarm for an important event (for example, a monthly meeting) or a specific task (for example, paying a utility bill), by using the `CalendarAlarm` interface. The alarm is triggered at the specified time to remind the user of the event or task.

- **Calendar change notifications**

  You can keep the calendar in your application synchronized with user-specific calendars, such as a calendar on a social networking Web site, by receiving notifications in your application when calendar items have been changed.

  The `addChangeListener()` method of the `Calendar` interface registers an event listener, which starts asynchronously when the `addChangeListener()` method returns the subscription identifier for the listener. You can use the `CalendarChangeCallback` interface to define listener event handlers for receiving the notifications.

  Every change made to the calendar database triggers an event for which you can define a notification. For batch mode operations, each operation generates only a single event. A recurring calendar event is treated as one event.

- **iCalendar 2.0 format conversions**

  You can convert the calendar items to the iCalendar format or back using the `CalendarEvent` object constructor and the `convertToString()` method of the `CalendarItem` interface, respectively.

  The conversion allows you to exchange calendar data between applications by sharing files with the `.ics` extension. The iCalendar format is independent of the underlying transport protocol, meaning that calendar items can be exchanged using a variety of protocols, including HTTP, SMTP, and Infrared.

  The iCalendar format can be used to store calendar item information and exchange calendar data over the Internet.

  The following example shows a sample event in the iCalendar format:

```
BEGIN: VCALENDAR
BEGIN: VEVENT
DTSTART: 20110714T150000Z
DTEND: 20110715T173000Z
```

```
SUMMARY: Team meeting
END: VEVENT
END: VCALENDAR
```

For more information about the Calendar API, see the Tizen Calendar API Reference.

Adding an Event to a Calendar

To add an event to the default system calendar:

1. Retrieve the default system calendar using the `getDefaultCalendar()` method of the `CalendarManager` interface. Specify the calendar type "EVENT" as a parameter.

```
var calendar = tizen.calendar.getDefaultCalendar("EVENT");
```

2. Create a `CalendarEvent` object and define the event properties:

```
var ev = new tizen.CalendarEvent(
{
   description:"HTML5 Introduction",
   summary:"HTML5 Webinar",
   startDate: new tizen.TZDate(2011, 3, 30, 10, 0),
   duration: new tizen.TimeDuration(1, "HOURS"),
   location:"Huesca",
```

3. To create a recurring event, define a recurrence rule. In this example, the event repeats once a day for three days.

```
   recurrenceRule: new tizen.CalendarRecurrenceRule("DAILY",
                                                    {occurrenceCount: 3})
});
```

4. To set up an alarm to remind about this event, create an alarm with the `CalendarAlarm` interface, and add the alarm to the event:

```
/* Alarm is triggered with sound 30 minutes before the event's start
time */
var alarm = new tizen.CalendarAlarm(new tizen.TimeDuration
                                    (30, "MINS"), "SOUND");
ev.alarms = [alarm];
```

5. Add the `CalendarEvent` object to the default calendar with the `add()` method of the `Calendar` object:

```
calendar.add(ev);  /* ev.id attribute is generated */
```

Note:

You can add a task item the same way as described above using the `"TASK"` calendar type and the `CalendarTask` constructor.

Adding Events to a Calendar in Batch Mode

To add multiple events to the default system calendar in batch mode:

1. Retrieve the default system calendar using the `getDefaultCalendar()` method of the `CalendarManager` interface:

```
var calendar = tizen.calendar.getDefaultCalendar("EVENT");
```

2. Define the items to be added as an array:

```
var ev = new tizen.CalendarEvent(
{
   description:"HTML5 Introduction",
   summary:"HTML5 Webinar",
   startDate: new tizen.TZDate(2011, 3, 30, 10, 0),
   duration: new tizen.TimeDuration(1, "HOURS"),
   location:"Huesca"
});
```

Note:

To keep the example as simple as possible, the array below contains only one event.

Note:

The `addBatch()` method is asynchronous. Normally, synchronous callbacks should be used to react to its success or failure.

3. Use the `addBatch()` method of the `Calendar` object to add the events in the array to the calendar:

```
calendar.addBatch([ev]);
```

Managing the Event

To manage a single calendar event:

1. Retrieve the default system calendar using the `getDefaultCalendar()` method of the `CalendarManager` interface. Specify the calendar type "EVENT" as a parameter.

```
var myCalendar = tizen.calendar.getDefaultCalendar("EVENT");
```

2. Retrieve all events stored in the calendar by using the `find()` method of the `Calendar` object:

```
myCalendar.find(eventSearchSuccessCallback)
```

Note:

To retrieve a specific set of events, you can specify a filter and sorting order for the search operation through the `filter` and `sortMode` parameters.

Note:

In this example, all the events are retrieved because no filter is used.

3. Update or delete the found event inside the `eventSearchSuccessCallback` event handler.

   In this example, the description parameter of the first event is changed and the event is updated in the calendar using the `update()` method. The second event is deleted using the `remove()` method.

```
/* Define the event success callback */
function eventSearchSuccessCallback(events)
{
  /* Update the first existing event */
  events[0].description = "New Description";
  myCalendar.update(events[0]);

  /* Delete the second existing event */
  myCalendar.remove(events[1].id);
}
```

Managing Multiple Calendar Events in Batch Mode

To manage multiple calendar events in batch mode:

1. Retrieve the default system calendar using the `getDefaultCalendar()` method of the `CalendarManager` interface. Specify the calendar type "EVENT" as a parameter.

```
var myCalendar = tizen.calendar.getDefaultCalendar("EVENT");
```

2. Retrieve all events stored in the calendar by using the `find()` method of the `Calendar` object:

```
myCalendar.find(eventSearchSuccessCallback);
```

Note:

To retrieve a specific set of events, you can specify a filter and sorting order for the search operation through the `filter` and `sortMode` parameters.

Note:

In this example, all the events are retrieved because no filter is used.

3. To update events:

   a. Define the items to be updated in the success event handler of the `find()` method:

```
function eventSearchSuccessCallback(events)
{
  events[0].description = "New Description 1";
  events[1].description = "New Description 2";
```

   b. Use the `updateBatch()` method to update multiple calendar items asynchronously:

```
  /* Update the first 2 existing events */
  myCalendar.updateBatch(events.slice(0, 2));
}
```

4. To delete events, use the `removeBatch()` method in the success event handler of the `find()` method to delete multiple calendar items asynchronously:

```
function eventSearchSuccessCallback(events)
{
  /* Delete the first 2 existing events */
  myCalendar.removeBatch([events[0].id, events[1].id]);
}
```

Updating a Recurring Calendar Event

To update a recurring calendar event:

1. Retrieve the default system calendar using the `getDefaultCalendar()` method of the `CalendarManager` interface. Retrieve the calendar event using the `get()` method by specifying the event ID.

```
var calendar = tizen.calendar.getDefaultCalendar("EVENT");
var event = calendar.get(evId);
```

2. Expand the recurring event to get its instances by using the `expandRecurrence()` method of the `CalendarEvent` object:

```
event.expandRecurrence(new tizen.TZDate(2012, 2, 1),
                       new tizen.TZDate(2012, 2, 15),
                       eventExpandSuccessCB);
```

   The expanded event instances have their own `id.uid` and `id.rid` attributes, where the `id.uid` attribute is the same for all instances.

3. Update a single instance of the expanded recurring event.

   In case of recurring events, you can use the second parameter of the `update()` method to determine, whether a single instance or all occurrences of the event are updated. If the parameter is set to `true`, all instances are updated, while if it is set to `false`, only the indicated instance of the recurring event is updated (based on the `id.rid` attribute).

   In this example, the second instance of the event is updated.

```
/* Success event handler */
function eventExpandSuccessCB(events)
{
  events[1].summary = 'updated summary';
  calendar.update(events[1], false);
}
```

Receiving Notifications on Calendar Changes

To receive notifications when calendar items are added, updated, or removed:

1. Define the needed variables:

```
/* Watcher identifier */
var watcherId = 0;

/* This example assumes that the calendar is initialized */
var calendar;
```

2. Define the event handlers for different notifications using the `CalendarChangeCallback` listener interface:

```
var watcher =
{
  /* When new items are added */
  onitemsadded: function(items)
  {
    console.log(items.length + " items were added");
  },

  /* When items are updated */
  onitemsupdated: function(items)
  {
    console.log(items.length + " items were updated");
  },

  /* When items are deleted */
  onitemsremoved: function(ids)
  {
    console.log(ids.length + " items were removed");
  }
};
```

3. Register the listener to use the defined event handlers:

```
watcherId = calendar.addChangeListener(watcher);
```

4. To stop the notifications, use the `removeChangeListener()` method:

```
function cancelWatch()
{
  calendar.removeChangeListener(watcherId);
}
```

Converting Calendar Items

The following examples show you how to make calendar item exchange more efficient in your application by converting calendar events to and from the iCalendar format:

- To convert an iCalendar string to a calendar event:
  a. Retrieve the default system calendar using the `getDefaultCalendar()` method of the `CalendarManager` interface. Specify the calendar type “EVENT” as a parameter.

```
var calendar = tizen.calendar.getDefaultCalendar("EVENT");
```

  b. Create a new `CalendarEvent` object from the iCalendar string and add it to the default calendar:

```
try
{
  var ev = new tizen.CalendarEvent
  ("BEGIN:VCALENDAR\r\n" +
   "BEGIN:VEVENT\r\n" +
   "DTSTAMP:19970901T1300Z\r\n" +
   "DTSTART:19970903T163000Z\r\n" +
   "DTEND:19970903T190000Z\r\n" +
   "SUMMARY:Annual Employee Review\r\n" +
```

```
    "CATEGORIES:BUSINESS,HUMAN RESOURCES\r\n" +
    "END:VEVENT\r\n" +
    "END:VCALENDAR", "ICALENDAR_20");

  calendar.add(ev);
  console.log('Event added with UID ' + ev.id.uid);
}
```

- To convert multiple iCalendar strings and import them to a calendar, convert the strings one by one and then use the `addBatch()` method to add all the events at once in batch mode.
- To convert a calendar event to the iCalendar format:
  a. Retrieve the default system calendar and find all calendar items which include the "Tizen" string in the summary attribute:

```
var myCalendar;

myCalendar = tizen.calendar.getDefaultCalendar("EVENT");

/* Define a filter */
var filter = new tizen.AttributeFilter("summary",
                                       "CONTAINS",
                                       "Tizen");

/* Search for the events */
myCalendar.find(eventSearchSuccessCallback, errorCallback, filter);
```

  b. Convert a calendar item to an iCalendar string in the success event handler of the `find()` method using the `convertToString()` method:

```
function eventSearchSuccessCallback(events)
{
  /* Convert the first event */
  var vevent = events[0].convertToString("ICALENDAR_20");
}
```

- To export and convert multiple events from a calendar, find the required events using the `find()` method with an applicable filter, and then convert the found events one by one.

Contact

The Contact API allows you to manage the contacts and persons listed in an address book. A contact object is always associated with a specific address book. A person object is an aggregation of one or more contacts associated with the same person.

Contact API Main Features

The main features of the Contact API include:

- **Address book management**

  To access a contact, you must first retrieve the `AddressBook` object. To access the device address books, you can use:

  – The `getDefaultAddressBook()` method of the `ContactManager` interface to retrieve the default address book

  - The `getAddressBooks()` method to retrieve all the available address books as an array
  - The `getAddressBook()` method to retrieve a specific address book

- **Contact management**

  You can add and manage contacts by using the applicable methods of the `AddressBook` interface. When managing a single contact at a time, the operations are handled in synchronously.

  You can create and manage multiple contacts simultaneously by using the applicable batch methods: `addBatch()`, `updateBatch()`, and `removeBatch()`. The batch mode provides faster, optimized processing of multiple contacts.

Note:

The batch mode does not provide progress information about operations. To ensure that you can view the progress, break the batch operation down into multiple smaller batch operations. For example, break down a batch of 100 update requests into 10 batch operations that update 10 records at a time. Breaking down a batch operation also helps you avoid blocking other database operations, such as add or remove.

  When searching for contacts, you can create attribute filters, attribute range filters, and composite filters based on specific filter attributes. You can also sort the search results.

- **Contact change notifications**

  You can keep the address book in your application synchronized with an external contact manager by receiving notifications in your application when contact information changes.

  The `addChangeListener()` method of the `AddressBook` interface registers an event listener. Event subscription starts automatically after the method is first called. The method returns the subscription identifier for the listener. You can use the `AddressBookChangeCallback` interface to define listener event handlers for receiving the notifications.

Note:

The listener object that is the first parameter of the `addChangeListener()` method must have at least one event handler defined. If no handlers are defined, a `TypeMismatchError` exception occurs.

  Every change made to the address book triggers an event for which you can define a notification. For batch mode operations, each operation generates only a single event.

- **Person management**

  You can manage persons, including searching, updating, and deleting, by using the applicable methods of the `ContactManager` interface. When managing a single person at a time, the operations are handled synchronously.

  You can manage multiple persons simultaneously using the applicable batch methods: `updateBatch()` and `removeBatch()`. The batch mode provides faster, optimized processing of multiple persons.

  A person is automatically added or modified when contacts are added to or unlinked from an existing person. You cannot add a person directly.

  When searching for a person, you can filter and sort the search results based on specific filter attributes.

- **vCard format conversions**

You can convert the contacts to vCard format or back to import and export contacts.

The vCard (RFC 2426) file format (`.vcf` or `.vcard`) is a standard for electronic business cards, which contain contact information, such as name, address, phone numbers, email addresses, URLs, logos, photographs, and audio clips.

The Contact API supports vCard version 3.0.

For more information about the Contact API, see the Tizen Contact API Reference.

Retrieving Address Books

To retrieve address books:

- To retrieve the default address book, use the `getDefaultAddressBook()` method of the `ContactManager` interface. The method returns the address book as an `AddressBook` object.

```
var myAddressbook;

/* Get the default address book */
myAddressbook = tizen.contact.getDefaultAddressBook();
```

- To retrieve all available address books, use the `getAddressBooks()` method. The method passes an array of `AddressBook` objects to the success event handler. The array contains all available address books on the device.

```
var addressBook;

function addressBooksCB(addressBooks)
{
  if (addressBooks.length > 0)
  {
    addressBook = addressBooks[0];
    console.log("The addressbook name is " + addressbook.name);
  }
}

/* Get the list of available address books */
tizen.contact.getAddressBooks(addressBooksCB);
```

All available address books on the device are retrieved.

- If you already know the ID of an address book, you can retrieve that specific address book with the `getAddressBook()` method.

Adding a Contact

To add a single contact to the default system address book:

1. Retrieve the default system address book using the `getDefaultAddressBook()` method of the `ContactManager` interface:

```
var addressbook = tizen.contact.getDefaultAddressBook();
```

2. Create a `Contact` object and define its properties as an object implementing the `ContactInit` interface (the parameter of the `Contact` constructor):

```
var contact = new tizen.Contact(
{
```

```
  name: new tizen.ContactName({firstName:"Jeffrey", lastName:"Hyman"}),
  emails: [new tizen.ContactEmailAddress("user@example.com")]
});
```

3. Add the `Contact` object to the default address book with the `add()` method of the `AddressBook` interface:

```
addressbook.add(contact);
```

Adding Multiple Contacts in Batch Mode

To add multiple contacts in batch mode:

1. Retrieve the default system address book using the `getDefaultAddressBook()` method of the `ContactManager` interface:

```
addressbook = tizen.contact.getDefaultAddressBook();
```

2. Define the items to be added as an array:

```
var c1 = new tizen.Contact(
{
  name: new tizen.ContactName({firstName:"Jeffrey", lastName:"Hyman"}),
  emails: [new tizen.ContactEmailAddress("user1@example.com")]
});

var c2 = new tizen.Contact(
{
  name: new tizen.ContactName({firstName:"Elton", lastName:"John"}),
  emails: [new tizen.ContactEmailAddress("user2@example.com")]
});
```

3. Use the `addBatch()` method of the `AddressBook` interface to add the contacts in the array to the address book:

```
addressbook.addBatch([c1, c2]);
```

Note:

The `addBatch()` method is asynchronous and callbacks should be used to react to its success or failure.

Managing a Contact

To manage a single contact:

- To retrieve a single contact, use the `get()` method of the `AddressBook` interface with the contact ID as a parameter. The following example uses an object of the `ContactRef` interface to retrieve the contact ID.

```
/* contactRef is retrieved by other APIs */
var contactRef;
try
{
  /* Retrieve the contact corresponding to the given reference */
  var addressBook = tizen.contact.getAddressBook(
```

```
                        contactRef.addressBookId);
  var contact = addressBook.get(contactRef.contactId);
}
```

- To update or delete a single contact:

  a. Retrieve the default address book using the `getDefaultAddressBook()` method of the `ContactManager` interface:

  ```
  var addressbook = tizen.contact.getDefaultAddressBook();
  ```

  b. Retrieve contacts stored in the address book by using the `find()` method of the `AddressBook` interface:

  ```
  var filter = new tizen.AttributeFilter("name.firstName", "CONTAINS",
                                         "Chris");
  var sortMode = new tizen.SortMode("name.lastName", "ASC");

  try
  {
    addressbook.find(contactsFoundCB, null, filter, sortMode);
  }
  ```

Note:

To retrieve a specific contact, you can specify a filter and sorting order for the search operation through the `filter` and `sortMode` parameters.

The contacts that match the filter are passed as an array to the registered success event handler in the selected sorting order.

  c. Update or delete the found contact inside the `contactsFoundCB` event handler.

  In this example, the first name of the first contact is changed and the contact is updated in the address book using the `update()` method. The second contact is deleted using the `remove()` method.

```
/* Define the event success callback */
function contactsFoundCB(contacts)
{
  contacts[0].name.firstName = "Christopher";
  try
  {
    /* Update the first found contact */
    addressbook.update(contacts[0]);

    /* Delete the second found contact */
    addressbook.remove(contacts[1].id);
  }
}
```

Managing Multiple Contacts in Batch Mode

To manage multiple contacts in batch mode:

1. Retrieve the default address book using the `getDefaultAddressBook()` method of the `ContactManager` interface:

```
var addressbook = tizen.contact.getDefaultAddressBook();
```

2. Retrieve contacts stored in the address book by using the `find()` method of the `AddressBook` interface:

```
var filter = new tizen.AttributeFilter("name.firstName", "CONTAINS",
                                       "Chris"};
var sortMode = new tizen.SortMode("name.lastName", "ASC");

try
{
  addressbook.find(contactsFoundCB, null, filter, sortMode);
}
```

3. To update contacts, define the contact changes to be made in the success event handler of the `find()` method:

```
function contactsFoundCB(contacts)
{
  /* Change the first names of all the found contacts */
  for (var i=0; i<contacts.length; i++)
  {
    contacts[i].name.firstName = "Christopher";
  }
```

4. Use the `updateBatch()` method to update multiple contacts asynchronously:

```
  /* Update all found contacts */
  addressbook.updateBatch(contacts);
}
```

5. To delete contacts, use the `removeBatch()` method in the success event handler of the `find()` method to delete multiple contacts asynchronously:

```
function contactsFoundCB(contacts)
{
 /* Delete the first 2 found contacts */
 addressbook.removeBatch([contacts[0].id, contacts[1].id]);
}
```

Receiving Notifications on Contact Changes

To receive notifications when contacts are added, updated, or removed:

1. Define the needed variables:

```
/* Watcher identifier */
var watcherId = 0;

/* This example assumes that the address book is initialized */
var addressbook;
```

2. Define the event handlers for different notifications using the `AddressBookChangeCallback` listener interface:

```
var watcher =
{
  /* When contacts are added */
  oncontactsadded: function(contacts)
```

```
  {
    console.log(contacts.length + " contacts were added");
  },

  /* When contacts are updated */
  oncontactsupdated: function(contacts)
  {
    console.log(contacts.length + " contacts were updated");
  },

  /* When contacts are deleted */
  oncontactsremoved: function(ids)
  {
    console.log(ids.length + " contacts were deleted");
  }
};
```

3. Register the listener to use the defined event handlers:

```
watcherId = addressbook.addChangeListener(watcher);
```

4. To stop the notifications, use the `removeChangeListener()` method of the `AddressBook` interface:

```
addressbook.removeChangeListener(watcherId);
```

Importing Contacts

To import a contact to the default system address book:

1. Retrieve the default system address book using the `getDefaultAddressBook()` method of the `ContactManager` interface:

```
var addressbook = tizen.contact.getDefaultAddressBook();
```

2. Create a new `Contact` object from the vCard string and add it to the default address book:

```
var contact = null;

try
{
  contact = new tizen.Contact
  ("BEGIN:VCARD\n"+
   "VERSION:3.0\n"+
   "N:Gump;Forrest\n"+
   "FN:Forrest Gump\n"+
   "ORG:Bubba Gump Shrimp Co.\n"+
   "TITLE:Shrimp Man\n"+
   "TEL;WORK:(111) 555-1212\n"+
   "TEL;HOME:(404) 555-1212\n"+
   "EMAIL;WORK;PREF:test@example.com\n"+
     "END:VCARD");

  addressbook.add(contact);
  console.log("Contact was added with ID " + contact.id);
}
```

To convert multiple strings and import them to an address book, convert the strings one by one and

then use the `addBatch()` method of the `AddressBook` interface to add all the contacts at once in batch mode.

Exporting Contacts

To export contacts from the default system address book:

1. Retrieve the default system address book using the `getDefaultAddressBook()` method of the `ContactManager` interface and get contacts from the address book using the `find()` method:

```
var addressbook;

var addressbook = tizen.contact.getDefaultAddressBook();

/* Define a filter */
var filter = new tizen.AttributeFilter("name.firstName", "CONTAINS",
                                       "Chris");

/* Search for the contacts */
addressbook.find(contactsFoundCB, errorCB, filter);
```

2. Convert each contact to a vCard string in the success event handler of the `find()` method.

   In the following example, the first found contact is exported by converting it to the vCard version 3.0 format:

```
function contactsFoundCB(contacts)
{
   /* Convert the first contact */
   var vcard = contacts[0].convertToString("VCARD_30");
}
```

Managing People

To manage a single person:

1. To retrieve people, use the `find()` method of the `ContactManager` interface:

```
tizen.contact.find(personsFoundCB);
```

2. Update or delete a found person in the `personsFoundCB()` event handler.

   In this example, the favorite flag of the first person is changed and the contact is updated using the `update()` method. The second person is deleted using the `remove()` method.

```
/* Define the event success callback */
function personsFoundCB(persons)
{
  persons[0].isFavorite = true;
  /* Update the first found person */
  tizen.contact.update(persons[0]);

  /* Delete the second found person */
  tizen.contact.remove(persons[1].id);
}
```

To merge multiple persons into a single person:

1. To retrieve people, use the `find()` method of the `ContactManager` interface:

```
tizen.contact.find(personsFoundCB);
```

2. Define the persons to be merged in the `personsFoundCB()` event handler:

```
function personsFoundCB(persons)
{
  var sourcePerson = persons[0];
  var targetPerson = persons[1];
```

3. Use the `link()` method to link contacts that are linked to the other person:

```
  /* Link 2 persons, contacts from sourcePerson are added to
     targetPerson and sourcePerson is removed */
  targetPerson.link(sourcePerson.id);
}
```

Messaging

The Messaging API allows you to use messaging functions for SMS, MMS, and email communication.

The Messaging API minimizes your coding efforts by providing one-step capabilities to perform all high-level messaging-related operations.

Messaging API Main Features

The main features of the Messaging API include:

- **Message writing**

  You can write a message using the `Message` object constructor, and you can set the message attributes and parameters using an object, which implements the `MessageInit` interface.

  You can add attachments to your MMS and email messages by creating `MessageAttachment` objects, defining the file path and the MIME type (image/png, text/pdf, or text/html) for each object, and assigning these objects as an array to the `attachments` attribute of a `Message` object.

  To save message drafts, use the `addDraftMessage()` method of the `MessageStorage` interface.

Note:

The system assigns a unique read-only message ID to each message the first time it is processed, such as when sending it or saving it as a draft.

- **Message sending**

  You can send messages using the `sendMessage()` method of the `MessageService` interface. The method requires both success and error event handlers. Depending on the result of the send operation, the message is moved to the device's Sent Items folder or Drafts folder and additionally stored in the message storage database.

- **Message management**

  You can find, update, and delete stored messages with the methods provided by the `MessageStorage` interface: `findMessages()`, `updateMessages()`, and `removeMessages()`.

When searching for messages, you can create attribute filters, attribute range filters, and composite filters based on specific filter attributes. You can also sort the search results.

- **Message storage change notifications**

  You can register event listeners to monitor changes in the message storage, a particular conversation, or a particular message folder.

  The `addMessagesChangeListener()`, `addConversationsChangeListener()`, and `addFoldersChangeListener()` methods of the `MessageStorage` interface register an event listener, which starts asynchronously once the method returns the subscription identifier for the listener. You can use the `MessagesChangeCallback`, `MessageConversationsChangeCallback`, and `MessageFoldersChangeCallback` interfaces to define listener event handlers for receiving notifications about the changes.

- **Server synchronization methods**

  You can load email messages and attachments from the email service with the `loadMessageBody()` and `loadMessageAttachment()` methods of the `MessageService` interface.

  To keep your email service accounts up-to-date, you can synchronize them with their respective external servers, such as Gmail and Microsoft Exchange, with the `sync()` method. You can also synchronize just one folder, such as the Inbox, with the `syncFolder()` method.

  You can specify the maximum number of messages that can be retrieved in each folder.

For more information about the Messaging API, see the Tizen Messaging API Reference.

Sending Messages

To send a message:

1. Retrieve the messaging service using the `getMessageServices()` method. The messaging service determines which message types you are handling (SMS, MMS, or email).

   In this example, the SMS service is retrieved:

```
tizen.messaging.getMessageServices("messaging.sms", serviceListCB,
                                   errorCallback);
```

2. In the success event handler of the `getMessageServices()` method, use the `Message` interface to define the content and attributes of the message, and then send the message using the `sendMessage()` method of the `MessageService` interface.

   If the message is not ready to be sent yet, save the message draft using the `addDraftMessage()` method of the `MessageStorage` interface.

```
function serviceListCB(services)
{
  /* Define SMS message */
  var msg = new tizen.Message("messaging.sms",
  {
  plainBody: "I will arrive in 10 minutes.",
    to: ["+34666666666", "+34888888888"]
  });
  /* Assume msgReady was defined */
  if (msgReady) {
     /* Send SMS message */
  services[0].sendMessage(msg, messageSent, messageFailed);
  }
```

```
  else
  {
    /* Save a draft */
    services[0].messageStorage.addDraftMessage(msg, successCallback,
                                               errorCallback);
  }
}
```

If you are sending MMS or email messages with attachments, add the attachments as an array of `MessageAttachment` objects:

```
var msg = new tizen.Message("messaging.email");
msg.attachments = [new tizen.MessageAttachment("images/myimage.png",
                                               "image/png"),
                   new tizen.MessageAttachment("documents/mydoc.pdf",
                                               "text/pdf")];
```

3. Define the message sending success event handler, which is called if the message is sent successfully. For emails, success means that the email was sent to the delivery system, not to final recipients of the email. For messaging technologies, such as SMS, where the message is sent individually to every message recipient, the success callback is invoked individually for each recipient.

```
function messageSent(recipients)
{
  for (var i = 0; i < recipients.length; i++)
  {
    console.log("The SMS has been sent to " + recipients[i]);
  }
}
```

Defining the `errorCallback` allows you to handle all possible errors and exceptions that can occur when the message delivery fails.

Managing Messages

To manage messages:

1. To retrieve messages whose sender is "me" from the message storage, use the `findMessages()` method of the `MessageStorage` interface with a filter:

```
function serviceListCB(services)
{
  emailService = services[0];
  /* Set the attribute filter */
  var filter = new tizen.AttributeFilter("from", "CONTAINS", "me");
  emailService.messageStorage.findMessages(filter, messageArrayCB,
                                           errorCallback);
}

tizen.messaging.getMessageServices("messaging.email", serviceListCB,
                                   errorCallback);
```

The `findMessages()` method returns an array of `Message` objects as the search result. The search result does not contain the actual bodies of the messages. To load a message body, use the `loadMessageBody()` method of the `MessageService` interface.

2. To update a message in the message storage, use the `updateMessages()` method. The method uses the array of `Message` objects retrieved by the `findMessages()` method in step 1.

   In this example, the `isRead` attribute of the first `Message` object in the array is updated to `true`:

```
function messageArrayCB(messages)
{
  messages[0].isRead = true;
  emailService.messageStorage.updateMessages(messages, successCallback,
                                              errorCallback);
}
```

3. To delete a message from the message storage, use the `removeMessages()` method:

```
function messageArrayCB(messages)
{
  emailService.messageStorage.removeMessages(messages, successCallback,
                                              errorCallback);
}
```

Synchronizing Emails

To synchronize emails:

1. Retrieve the messaging service using the `getMessageServices()` method:

```
tizen.messaging.getMessageServices("messaging.email", serviceListCB,
                                   errorCallback);
```

2. Search for all email messages with attachments using the `findMessages()` method of the `MessageStorage` interface:

```
service.messageStorage.findMessages(
          new tizen.AttributeFilter("hasAttachment", "EXACTLY", true),
          messageQueryCallback);
```

3. To load a message body, use the `loadMessageBody()` method of the `MessageService` interface:

```
/* Success callback for the search operation */
function messageQueryCallback(messages)
{
  for (var i = 0; i < messages.length; i++) {
    var message = messages[i];
    if (!message.body.loaded) {
      tizen.messaging.loadMessageBody(message, successCallback,
                                       errorCallback);
```

4. To download the message attachments, use the `loadMessageAttachment()` method with an array of attachments (with valid file paths) as a parameter:

```
      tizen.messaging.loadMessageAttachment(message.attachments[0],
                                            successCallback,
                                            errorCallback);

    }
  }
}
```

5. To synchronize the messages with an external server:

- To synchronize all account folders, use the `sync()` method:

```
/* Synchronize the folders in the success event handler */
function servicesListSuccessCB(services)
{
  services[0].sync(serviceSyncedCB, null, 30);
}
/* Get the email service */
tizen.messaging.getMessageServices("messaging.email",
                                   servicesListSuccessCB);
```

- To synchronize a specific folder, use the `syncFolder()` method.

  In this example, only folders of the "INBOX" type are synchronized:

```
var emailService; /* Assume email service is initialized */
function serviceCallback(services)
{
  emailService = services[0];
}

/* Synchronize in the search success event handler */
function folderQueryCallback(folders)
{
  for (var i = 0; i < folders.length; i++) {
    if (folders[i].type === "INBOX") {
      emailService.syncFolder (folders[i], folderSyncedCB, null, 30);
    }
  }
}

/* Get the email service */
tizen.messaging.getMessageServices("messaging.email",
                                   serviceCallback, errorCallback);

/* Search for specific folders */
var filter = new tizen.AttributeFilter("serviceId", "EXISTS");

emailService.messageStorage.findFolders(filter,
                                        folderQueryCallback));
```

Receiving Notifications on Message Storage Changes

To receive notifications when messages and message folders are added, updated, or removed:

1. Define the needed variable:

```
/* Watch identifier */
var watchId;
```

2. Define the event handlers for different notifications by implementing the `MessagesChangeCallback` listener interface:

```
var messageChangeCallback =
{
  /* When messages are updated */
  messagesupdated: function(messages)
  {
```

```
    console.log(messages.length + " message(s) updated");
  },

  /* When messages are added */
  messagesadded: function(messages)
  {
    console.log(messages.length + " message(s) added");
  },

  /* When messages are deleted */
  messagesremoved: function(messages)
  {
    console.log(messages.length + " message(s) removed");
  }
};
```

3. Register the listener to use the defined event handlers:

```
watchId = msgService.messageStorage.addMessagesChangeListener(
            messageChangeCallback);
```

4. To stop receiving the notifications, use the `removeChangeListener()` method of the `MessageStorage` interface:

```
msgService.messageStorage.removeChangeListener(watchId);
```

Note:

To provide notifications for changes in specific conversations or message folders, use the applicable methods and event handlers similarly as above.

Multimedia

Tizen enables you to search for content (images, videos, music or other) located on the local device storage, using the Content API. You can also perform content management tasks, such as viewing and updating content attributes.

Discovering Content

The Tizen system maintains a database of available multimedia content. This database allows fast searches for media files and assigning additional information to content, such as marking a video clip as a favorite.

The Content API allows you to search and update the database. The `ContentManager` interface provides the `find()` method for retrieving information about media files.

Each multimedia file is represented as an instance of one of the following `Content` interfaces:

- `AudioContent` represents audio files.
- `ImageContent` represents image files.
- `VideoContent` represents video files.

Listing All Audio Files

To retrieve a list of all audio files in the system:

1. Define the callback methods for the search. The `findSuccess()` callback receives a list of `Content` objects and shows the title of each object.

```
function findSuccess(items)
{
  console.log("Found " + items.length + " audio tracks:");
  for(var i=0; i<items.length; i++)
  {
    console.log(i.toFixed() + ". " + items[i].title + " (" +
      items[i].name + ")");
  }
}
function findError(err)
{
   console.log("Error: " + err.message);
}
```

2. To retrieve the audio files, use the `find()` method of the `ContentManager` interface with an attribute filter for limiting the search to only audio files.

   The third parameter of the `find()` method allows you to limit the search to a single folder. In this example, use the `null` value to search all folders.

```
var audioOnly = new tizen.AttributeFilter("type", "EXACTLY", "AUDIO");

tizen.content.find(findSuccess, findError, null, audioOnly);
```

Other types of media content can be discovered in a similar way:

```
var videoOnly = new tizen.AttributeFilter("type", "EXACTLY", "VIDEO");

var imagesOnly = new tizen.AttributeFilter("type", "EXACTLY", "IMAGE");
```

Discovering New Added Content

The Content API also allows you to receive notifications about changes in the content database. For example, to receive notifications when new content is added, set an event listener using the `setChangeListener()` method of the `ContentManager` interface:

```
tizen.content.setChangeListener(
{
   oncontentadded: function(item)
   {
      console.log("New " + item.type + " detected: " + item.contentURI);
   }
});
```

To stop receiving notifications, use the `unsetChangeListener()` method:

```
tizen.content.unsetChangeListener();
```

Capturing Images and Video

The Web APIs do not provide direct access to the device camera, but you can launch the native

Camera application through the Tizen Application Manager.

The Tizen Application Manager allows applications to use other applications to perform specific operations, such as capture images and video with the device camera. You can access the Tizen Application Manager with the Application API. To use the Application API, your application must have the following permission defined in the `confix.xml` file:

`https://tizen.org/privilege/application.launch`

To launch another application, you must create an `ApplicationControl` object describing the operation you want to perform. The `ApplicationControl` object is necessary for calling the `launchAppControl()` method, which launches the other application.

You can run any system application by specifying its ID. To launch a specific application, define its ID as the second parameter of the `launchAppControl()` method. You can also allow the system to decide which application is best-suited to performing the requested operation.

The `launchAppControl()` method is asynchronous, meaning its results are available through a callback mechanism. Define the success and error callbacks as the third and fourth parameters of the `launchAppControl()` method, respectively. The success callback is triggered when the application is launched successfully. Otherwise, the error callback is triggered. To receive the result of the request operation, define the reply callback as the fifth parameter.

For more information about the Application API, see the Tizen Application API Reference.

Taking a Photo

To take a photo with the device camera:

1. Define the following permission in the `config.xml` file:

```
<tizen:privilege name="http://tizen.org/privilege/application.launch" />
```

2. Create an `ApplicationControl` object for the `http://tizen.org/appcontrol/operation/create_content` operation, with the MIME type set to `image/jpg`:

```
var appControl = new tizen.ApplicationControl(
      "http://tizen.org/appcontrol/operation/create_content",
      null,
      "image/jpg");
```

3. Define the success, error, and reply callbacks. The reply callback must implement the `ApplicationControlDataArrayReplyCallback` interface.

```
function successCb()
{
  console.log("Camera application launched successfully");
}
function errCb(err)
{
  console.log("Error: " + err.message);
}
var replyCB =
{
   onsuccess: function(pairs)
   {
      for(var i=0; i<pairs.length; i++)
      {
         if(pairs[i].key ===
         "http://tizen.org/appcontrol/data/selected")
```

```
            {
                console.log("picture taken: " + pairs[i].value[0]);
            }
        }
    },
    onfailure: function()
    {
        console.log("FAILED");
    }
};
```

4. Request the system to run the camera application by calling the `launchAppControl()` method:

```
tizen.application.launchAppControl(appControl, null, successCb, errCb,
replyCB)
```

Playing Audio and Video

Tizen supports the HTML5 audio and video elements, which can be used to play multimedia files and streams without a separate plug-in.

Using JavaScript, the playback can be controlled with media events. The audio and video elements used as media elements inherit all the properties and methods of the `HTMLMediaElement` interface.

The main features of the audio and video elements with JavaScript include:

- **Creating a player**

  You can create a simple audio and video player.

- **Controlling playback**

  You can use the `play()` and `pause()` methods of the `Media` object to control the playing and pausing of media files. With media events, additional features can be used.

- **Retrieving duration and play time**

  You can retrieve the duration and play time of the media file if its metadata (such as playing time, duration, and video width and height) is loaded.

- **Playing from a random position**

  You can indicate the playback time by playing the media file from a random position. To do this, you must change the `currentTime` property value of the `Media` object to trigger the `timeupdate` event.

- **Retrieving progress state**

  You can retrieve and display the download progress state using the `Progress` media event, which is triggered when information related to the progress of a media object loading media contents is updated.

- **Checking supported media formats**

  You can check whether the media data can be played by using the `canPlayType()` method. The MIME type must be set in the Web server in a format that is supported in Tizen. If a non-supported MIME type is used, you can handle exceptions in advance.

  Tizen supports the following media codecs:

  - For audio: AAC, AMR-NB, AMR-WB, MP3, Vorbis
  - For video: H.263, H.264, MPEG-4, Theora

For more information about the HTML5 audio and video elements, see the W3C HTML5 specification.

Creating an Audio and Video Player

To create a simple HTML5 player for streaming audio and video:

1. To create an audio player, create an audio element including the necessary attributes:

```
<audio id="audio" src="media/audio_sample.mp3"
       preload="auto" controls>
</audio>
```

2. To create a video player, create a video element including the necessary attributes. In addition to the attributes available for the audio element, you can also use the `width`, `height`, and `poster` attributes.

```
<video id="video" src="media/video_sample.mp4"
       width="400" height="220" poster="media/poster_sample.png"
       preload="auto" controls>
</video>
```

A player with a play control (built-in control provided in the browser) is created. The control is visible only when the `controls` attribute is added. If the `poster` attribute is not defined, the first frame of video is shown on the screen before playback.

Note:

The `preload` attribute is set to `auto` by default, meaning that the media metadata is automatically loaded. If you do not want to load the metadata, set the attribute value to `metadata` or `none`.

Note:

Carefully consider user experience before using the autoplay feature, which plays content automatically without user interaction. In a mobile network, the user can incur unintended Internet packet fees or interfering factors, such as playback being stopped unintentionally.

Playing Media Files

To play and pause media files using custom controls:

1. Create the video element and buttons used to control the play and pause actions:

```
<div class="media">
  <video id="video" src="media/video_sample.mp4"></video>
  <div>
    <button id="v-play" type="button">play</button>
    <button id="v-pause" type="button" disabled>pause</button>
  </div>
</div>
```

   The **Pause** button is disabled until the play event occurs.

2. Define the button functions. Play and pause the media file using the `play()` and `pause()` methods of the `HTMLMediaElement` interface.

```
<script>
  var play_button = document.getElementById("v-play");
  var pause_button = document.getElementById("v-pause");
```

```
var video = document.getElementById("video");

play_button.addEventListener("click", function()
{
  video.play();  /* Play movie */
}, false);

pause_button.addEventListener("click", function()
{
  video.pause(); /* Pause movie */
}, false);

video.addEventListener("play", function()
{
  pause_button.disabled = false;
}
```

Streaming Multimedia

You can access multimedia streams from local devices, such as the video camera and microphone. This functionality is based on the W3C `getUserMedia()` specification.

The `getUserMedia()` specification is a work in progress and not fully standardized. Many browser projects have implemented their own versions of the specification. The Tizen platform uses WebKit and provides access to local multimedia streams through the `webkitGetUserMedia()` method.

Note:

The implementation of the `getUserMedia()` specification is very much a bleeding-edge technology. Any APIs or functionality discussed in this section are liable to change in future versions of the specification and may work differently in later versions of Tizen.

For more information about media streams, see the W3C Media Capture and Streams specification.

This specification defines a URL interface that provides the `createObjectURL()` method, which you can use to create a Blob URL from a media stream object. As with the `getUserMedia()` method, the WebKit project has implemented its own version of the `createObjectURL()` method and provides a `webkitURL` interface containing the `createObjectURL()` method.

The `createObjectURL()` method takes the media stream object generated by the `webkitGetUserMedia()` method and creates a URL that is suitable as an input for the video element's source attribute.

Streaming Video from the Device Camera

To access a video stream:

1. Create the HTML5 video element and a button for controlling video stream access:

```
<body>
  <video id="videoPlay" src="" autoplay controls></video>
  <br />
  <input type="button" value="START" onclick="getVideoStream();"
         id="btnStart">
</body>
```

2. Use the `navigator.webkitGetUserMedia()` method to access the video stream:

```
<script>
  function getVideoStream()
  {
    navigator.webkitGetUserMedia({video: true}, successCallBack,
                                 errorCallBack);
  }
</script>
```

   The first parameter is mandatory and assigns a JSON object to determine which media element (audio or video) to use.

   The system asks the user for permission to provide the application access to the camera video stream.

3. Retrieve the video stream information, create a stream URL, and attach the video stream to the video element:

```
<script>
  function successCallback(stream)
  {
    var URL = window.webkitURL;
    document.getElementById("videoPlay").src =
      URL.createObjectURL(stream);
  }
</script>
```

NFC

The NFC API allows you to use the Near Field Communication (NFC) service for exchanging data between NFC devices ("peers") or tags. The devices can share contacts, photos, and videos, and can also act as smart cards. You can use an NFC device to communicate with NFC tags for a range of activities, such as paying bills or downloading coupons.

NFC provides the following advantages over other short-range communication technologies:

- Faster setup
- Lower power consumption
- No device pairing requirements
- Reduction of unwanted interruptions

An NFC tag is a chip that can securely store personal information, such as debit card numbers or contact details. You can use the methods of the `NFCTag` interface to access an NFC tag for reading or writing information. NFC tag types are identified using the type attribute of the `NFCTagType` type definition.

Note:

Tizen supports the following NFC tag types: GENERIC_TARGET, ISO14443_A, ISO14443_4A, ISO14443_3A, MIFARE_MINI, MIFARE_1K, MIFARE_4K, MIFARE_ULTRA, MIFARE_DESFIRE, ISO14443_B, ISO14443_4B, ISO14443_BPRIME, FELICA, JEWEL, and ISO15693.

The NFC forum defines the NFC data exchange format (NDEF) for encapsulating the data exchanged between two NFC devices or an NFC device and an NFC tag. An NDEF message can store data in various formats, such as text, MIME type objects, or ultra-short records based on the RTD (Record

Type Definition) specification. NFC tags use NDEF for exchanging messages.

NFC API Main Features

The main features of the NFC API include:

- **NFC device management**

  You can manage NFC connectivity by enabling or disabling the NFC service.

  To use NFC, retrieve the default NFC adapter using the `getDefaultAdapter()` method of the `NFCManager` interface. You can enable and disable NFC using the `setPowered()` method.

- **NFC tag and peer detection**

  To receive notifications when an NFC tag or peer device has been detected use the `setTagListener()` and `setPeerListener()` methods of the `NFCAdapter` interface. These methods register event listeners, which trigger notifications when an NFC tag or peer device is detected, respectively. You can use the `NFCTagDetectCallback` and `NFCPeerDetectCallback` interfaces to define event handlers for receiving the notifications about attaching and detaching NFC tags and peers, respectively.

- **NDEF message manipulation**

  You can handle NDEF messages by first creating NDEF records, using the `NDEFRecord` constructor (you can also use `NDEFRecordText`, `NDEFRecordURI`, and `NDEFRecordMedia` interfaces), and then adding the records to an NDEF message using the records attribute of the `NDEFMessage` interface.

- **NDEF data exchange**

  You can exchange NDEF data between tags and peers. To exchange data between tags, use the `readNDEF()` and `writeNDEF()` methods of the `NFCTag` interface.

  To exchange data between peers, use the `sendNDEF()` method to send messages and the `setReceiveNDEFListener()` method of the `NFCPeer` interface to register an event listener, which triggers an event when an NDEF message is received from a peer.

  You can use the `NDEFMessageReadCallback` interface to define event handlers for reading NDEF messages from tags and peer devices.

> **Note:**
>
> If an application is in the background (hidden) and uses the `writeNDEF()`, `transceive()`, or `sendNDEF()` method, an error callback is launched. These methods can only be used while the application is visible.

The NFC service can launch an NFC application based on the NDEF message content using the application control functionalities. If the application is designed to be launched this way, it must have the following operation defined in the `config.xml` file:

```
http://tizen.org/appcontrol/operation/nfc/wellknown
```

When the NFC device (powered on) reads an NFC tag or receives an NDEF message with the record type of the first record (`NDEFRecord`) set to `NFC_RECORD_TNF_WELL_KNOWN`, the NFC application is launched.

NFC applications can also be launched by card emulation transactions. NFC devices can communicate with point of sale (POS) terminals using the card emulation functionality to, for example, make a payment. If the application control with the

`http://tizen.org/appcontrol/operation/nfc/transaction` operation is defined in the `config.xml` file and a transaction caused by the card emulation functionality occurs, the NFC application is launched.

The following table lists the NFC operations, schemes, and MIME types.

Table 9: NFC operations

Operation	Scheme	MIME
`http://tizen.org/appcontrol/operation/nfc/empty`	NULL	NULL
`http://tizen.org/appcontrol/operation/nfc/wellknown`	`<scheme>:<host>/<path>` URL examples: • `http` • `http://tizen.org/` • `http://tizen.org/about/devices` • `http://tizen.org/about/*` URN examples: • `tel` • `mailto` • `mailto:tommy@tizen.org` The `<protocol_code>` and `<scheme>` must match. For more information, see the NFCForum-TS-RTD_URI_1.0 and NFC RTD (Record Type Definition) documentation on the NFC forum.	`U/<protocol_code>` For example: • `U/0x03` • `U/0x05` • `U/*`
	NULL	`<type_string>/*` For example: • `sp/*` • `T/*` • `*/*`
`http://tizen.org/appcontrol/operation/nfc/mime`	NULL	`<type_string>/<subtype_string>` (case-insensitive) For example: • `text/x-vard` • `text/*` • `*/*`
`http://tizen.org/appcontrol/operation/nfc/uri`	`<uri>` For example: • `http://tizen.org/about/devices`	NULL
`http://tizen.org/appcontrol/operation/nfc/external`	`<scheme>:<string>` (case-insensitive) For example:	NULL

	• nfc:ext.tizen.org.ABC	
http://tizen.org/appcontrol/operation/nfc/transaction	nfc://secure/<SE name>/aid/<aid> For example: • nfc://secure/SIM1/aid/123456789 • nfc://secure/SIM1/aid/1234* • nfc://secure/SIM1/aid/*	NULL

For more information about the NFC API, see the Tizen NFC API Reference.

Managing NFC Connectivity

To manage NFC connectivity:

1. To access the NFC adapter, use the `getDefaultAdapter()` method:

```
var nfcAdapter = tizen.nfc.getDefaultAdapter();
```

2. To enable NFC, use the `setPowered()` method of the `NFCAdapter` interface with the first parameter set to true:

```
nfcAdapter.setPowered(true, onPowerOn, onPowerOnFails);
```

3. To disable NFC, use the `setPowered()` method with the first parameter set to false:

```
nfcAdapter.setPowered(false);
```

Detecting NFC Tags and Peer Devices

To detect NFC tags and peer devices:

1. To access the NFC adapter, use the `getDefaultAdapter()` method:

```
var nfcAdapter = tizen.nfc.getDefaultAdapter();
```

2. Define the event handlers for NFC tag detection using the `NFCTagDetectCallback` listener interface:

```
var setTagDetect =
{
  /* When an NFC tag is detected */
  onattach: function(nfcTag)
  {
    console.log("NFC Tag detected. Its type is: " + nfcTag.type);
  }

  /* When an NFC tag becomes unavailable */
  ondetach: function()
  {
    console.log("NFC Tag unavailable");
  }
}
```

3. Register the listener to use the defined event handlers.

You can limit the listener to detect only specific NFC tag types by defining the tag types as the second parameter of the `setTagListener()` method. In this example, only MIFARE tags are detected:

```
/* Defines the tag types to be detected */
var tagFilter = ["MIFARE_MINI", "MIFARE_1K", "MIFARE_4K",
                 "MIFARE_ULTRA", "MIFARE_DESFIRE"];

/* Registers the event listener */
nfcAdapter.setTagListener(setTagDetect, tagFilter);
```

4. To stop the tag detection, use the `unsetTagListener()` method:

```
nfcAdapter.unsetTagListener();
```

NFC peers are detected similarly as NFC tags, except that the `setPeerListener()` method is used to register the `NFCPeerDetectCallback` listener interface, and the `unsetPeerListener()` method is used to stop the peer detection.

Handling NDEF Messages

To handle an NDEF message:

1. To create an NDEF URI record, create an `NDEFRecordURI` interface instance and specify the URI parameter:

```
var newRecord = new tizen.NDEFRecordURI("https://www.tizen.org/");
```

   You can also create instances of the `NDEFRecord`, `NDEFRecordText`, and `NDEFRecordMedia` interfaces based on the record type to be created.

2. Create an `NDEFMessage` interface instance:

```
var newMessage = new tizen.NDEFMessage();
```

3. To add an NDEF record to the NDEF message, use the `records` attribute of the `NDEFMessage` interface:

```
newMessage.records[0] = newRecord;
```

Exchanging NDEF Data with NFC Tags

To exchange NDEF data with an NFC tag:

1. To read data from an NFC tag, use the `readNDEF()` method of the `NFCTag` interface. This method retrieves the NDEF messages saved on the NFC tag and passes them to the `NDEFMessageReadCallback` listener.

```
/* NDEFMessageReadCallback listener */
function readMessage(message)
{
  console.log("Record Count is " + message.recordCount);
}

/* Check whether the NFC tag supports NDEF format */
if (Tag.isSupportedNDEF)
{
  /* Read NDEF data */
```

```
  Tag.readNDEF(readMessage);
}
```

2. To write data to an NFC tag, use the `writeNDEF()` method:

```
var newMessage = new tizen.NDEFMessage();
function writeCallback()
{
  console.log("Success!");
}
Tag.writeNDEF(newMessage, writeCallback);
```

3. You can use the `transceive()` method to transfer raw data, as a byte array, to an NFC tag. But sending data to the NFC tag this way requires knowledge of the underlying details of the tag.

Exchanging NDEF Data with Peer Devices

To exchange NDEF data with a peer device:

1. To receive NDEF messages from a peer device, use the `setReceiveNDEFListener()` method of the `NFCPeer` interface. This method registers the `NDEFMessageReadCallback` listener interface, which is invoked when an NDEF message from a peer device is read.

```
/* NDEFMessageReadCallback listener */
function readMessage(message)
{
  console.log("Record Count is " + message.recordCount);
}

/* Set a listener to receive an NDEF message */
peer.setReceiveNDEFListener(readMessage);
```

2. To send an NDEF message to a peer device, use the `sendNDEF()` method:

```
var newMessage = new tizen.NDEFMessage();
peer.sendNDEF(newMessage);
```