



Mentor® Embedded Inflexion® UI Runtime for ReadyStart™

User and Reference Manual

Software Version 2.6.0

April 2013

© 2013 Mentor Graphics Corporation
All rights reserved.

This document contains information that is proprietary to Mentor Graphics Corporation. The original recipient of this document may duplicate this document in whole or in part for internal business purposes only, provided that this entire notice appears in all copies. In duplicating any part of this document, the recipient agrees to make every reasonable effort to prevent the unauthorized use and distribution of the proprietary information.

This document is for information and instruction purposes. Mentor Graphics reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should, in all cases, consult Mentor Graphics to determine whether any changes have been made.

The terms and conditions governing the sale and licensing of Mentor Graphics products are set forth in written agreements between Mentor Graphics and its customers. No representation or other affirmation of fact contained in this publication shall be deemed to be a warranty or give rise to any liability of Mentor Graphics whatsoever.

MENTOR GRAPHICS MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

MENTOR GRAPHICS SHALL NOT BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATED TO THIS PUBLICATION OR THE INFORMATION CONTAINED IN IT, EVEN IF MENTOR GRAPHICS CORPORATION HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

RESTRICTED RIGHTS LEGEND 03/97

U.S. Government Restricted Rights. The SOFTWARE and documentation have been developed entirely at private expense and are commercial computer software provided with restricted rights. Use, duplication or disclosure by the U.S. Government or a U.S. Government subcontractor is subject to the restrictions set forth in the license agreement provided with the software pursuant to DFARS 227.7202-3(a) or as set forth in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clause at FAR 52.227-19, as applicable.

Contractor/manufacturer is:

Mentor Graphics Corporation

8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777.

Telephone: 503.685.7000

Toll-Free Telephone: 800.592.2210

Website: www.mentor.com

SupportNet: supportnet.mentor.com/

Send Feedback on Documentation: supportnet.mentor.com/doc_feedback_form

TRADEMARKS: The trademarks, logos and service marks ("Marks") used herein are the property of Mentor Graphics Corporation or other third parties. No one is permitted to use these Marks without the prior written consent of Mentor Graphics or the respective third-party owner. The use herein of a third-party Mark is not an attempt to indicate Mentor Graphics as a source of a product, but is intended to indicate a product from, or associated with, a particular third party. A current list of Mentor Graphics' trademarks may be viewed at: www.mentor.com/trademarks.

Table of Contents

Chapter 1

Introduction	13
Mentor Embedded Inflexion UI Runtime	13
Included Components	13
Software Overview	14
Inflexion UI Runtime Architecture	14
Packages	17
Engine API Overview	17
Porting Layer Overview	18
Integration Layer Overview	18
API References	19

Chapter 2

Getting Started	21
Initializing Mentor Embedded Inflexion UI Runtime Component	21
Building the Mentor Embedded Inflexion UI Runtime Package	21
Including the Mentor Embedded Inflexion UI SimpleEdit Integrated Module Component	22
Including the Mentor Embedded Inflexion UI Porting Layer Package	22
Including the Freetype Package	22

Chapter 3

Configuration	23
File System Support	23
use_rom_files	23
use_platform_files	23
Rendering Mode	24
render_internal	24
render_internal_compressed	24
render_direct_opengl	24
render_direct_opengl_20	24
Canvas Mode	25
canvas_mode_8888	25
canvas_mode_888	25
canvas_mode_565	25
canvas_mode_1555	25
canvas_mode_444	25
Canvas Order	26
canvas_order_rgb	26
Open GL Render-mode-specific Options	26
bpp	26
depth_buffer_size	26
swap_interval	26

msaa_samples	27
Performance	27
use_big_endian	27
Benchmarking Support	27
enable_bm_frame_rate	27
enable_bm_heap_usage	27
Feature Support	28
use_stylus	28
use_mouseover	28
use_bitmapped_fonts	28
use_native_fonts	28
use_plugin_elements	28
validate_projects	29
use_deferred_component_loading	29
Development Support	29
debug_mode	29
Porting Layer Configuration	29
a_drive_map	29
b_drive_map	29
c_drive_map	30
default_font_header_file_name	30
 Chapter 4	
Evaluating Performance and Benchmarking.....	31
Configuring the Engine	31
Configuring the Porting Layer and the Demonstration	31
Configuring the Averaging of the Frame Times	33
Outputting the Frame Rate Readings	33
Benchmarking and Testing Guidelines	34
Benchmarking Tips	34
Testing Framework	34
 Chapter 5	
About Modules	37
Module Creation	38
Module Use	38
Advice for Module Writers	39
Modules and IFXM Functions	39
IFXM Function Formats	39
Modules Supporting Links	40
Modules Supporting Field Tokens	41
Modules Supporting Dynamic Menus	43
Modules Supporting Integrated Applications	48
Module Definition File	51
Format	51
Fields	52
Links	53
Menus	54

Table of Contents

Elements	54
Triggers	55
SimpleEdit Module Example	55
Device Definition File	56
Portable Key Codes	56
Format	57
Graphical Capabilities	57
Displays	57
Keys	58
Languages	58
Display Mode	58
Design Sizes	58
Target-Specific Tool Definitions	59
Device Definition File Example	60
Application Definition File	62
Format	62
supportedModules	63
entryPoints	63
Application Definition File Example	64
Mentor Embedded Inflexion UI Integrator	64
Usage	64
Integrator Example	65
 Chapter 6	
Porting Overview	67
Render Modes	67
Buffered Internal	67
Buffered Internal Compressed	68
Porting Layer	68
General	68
Memory	69
Canvas Management and Rendering	69
Native Fonts	73
Mutex Support	73
File System	73
Benchmarking	74
Custom Image Format Support	74
 Chapter 7	
Modules	75
SimpleEdit Module	75
SimpleEdit Module Definition File	75
SimpleEdit Module Header File	76
Module Implementation	76
Battery Module	77
Module Header	77
Battery Module Implementation	78
Demo Module	78

Demo Module Definition File	78
Demo Module Header	79
Demo Module Implementation	79
Plugin Module	79
Plugin Module Definition File	79
Plugin Module Header	80
Plugin Module Implementation	80
Chapter 8	
Engine API Reference	81
Engine API (IFXE)	81
Types	81
List of Functions	81
Startup/Shutdown	82
IFXE_Initialize	83
IFXE_Shutdown	86
IFXE_Save_State	87
IFXE_Suspend	88
IFXE_Resume	89
Framework	90
IFXE_Process_Timers	91
IFXE_Set_Configuration	93
IFXE_Get_Configuration	95
IFXE_Refresh_All	96
IFXE_Refresh_Display	97
IFXE_Resume_Static_Animations	98
IFXE_Set_Idle	99
Input	100
IFXE_Key_Down	101
IFXE_Key_Up	103
IFXE_Touch_Up	104
IFXE_Touch_Down	105
IFXE_Touch_Move	106
File Access	107
IFXE_File_Open	108
IFXE_File_Read	109
IFXE_File_Seek	110
IFXE_File_Size	111
IFXE_File_Close	112
IFXI_RequestResourceSearch	113
Chapter 9	
Module Integration API Reference	115
C Language Modules	115
General Types	116
List of Methods	120
Module Framework	122
IFXM - Initialize	123

Table of Contents

IFXM - Shutdown	125
Menus	126
IFXM - OpenMenu	127
IFXM - CloseMenu	129
IFXM - GetItemCount	130
IFXM - GetMenuType	131
IFXM - GetLoadOnDemand	133
IFXM - GetPlaceholderSize	135
IFXM - GetPlaceholderData	136
IFXM - GetSortFieldSize	137
IFXM - GetSortFieldData	139
IFXM - GetSortDirection	141
IFXM - GetSortType	142
IFXM - GetFirstActiveItem	143
IFXM - SetActiveItem	144
IFXI_RequestMenuRefresh	145
IFXI_RequestQueryActiveItem	146
IFXI_RequestSetActiveItem	148
Field Data	149
IFXM - GetFieldStringSize	150
IFXM - GetFieldData	153
IFXM - SetFieldData	156
IFXI_RequestFieldRefresh	159
Plugin	161
IFXM - CreateElement	162
IFXM - DestroyElement	166
IFXM - ChangeElementMode	168
IFXM - ActivateElement	171
IFXM - DeactivateElement	173
IFXM - SetElementFocus	175
IFXM - UnsetElementFocus	177
IFXM - ProcessElementKeyDownEvent	178
IFXM - ProcessElementKeyUpEvent	180
IFXM - ProcessElementStylusDragEvent	182
IFXM - ProcessElementStylusCancelEvent	184
IFXM - ProcessElementStylusDownEvent	185
IFXM - ProcessElementStylusUpEvent	187
IFXM - GetElementCaretPosition	189
IFXM - PositionElement	191
IFXM - PaintElement	193
IFXI_RequestRefreshBufferedElement	195
IFXI_RequestResizeBufferedElementBuffer	196
IFXI_RequestFullScreenStart	198
IFXI_RequestFullScreenStop	199
Functions	200
IFXM - ExecuteLink	201
IFXE_Async_Link_Complete	203
IFXI_RequestExecuteLink	204
IFXI_RequestTriggerKey	205

Exclusivity	206
IFXM - ExclusivityStatusChange	207
IFXI_ReleaseExclusivity	208
IFXI_RequestExclusivity	209
Shared Buffers	210
IFXM - BufferCreateString	211
IFXM - BufferDestroy	213
IFXM - BufferSetFocus	215
IFXM - BufferUnsetFocus	216
IFXI_BufferCreateString	217
IFXI_BufferDestroy	219
IFXI_BufferSetFocus	220
IFXI_BufferUnsetFocus	221
Chapter 10	
Porting API Reference	223
About Type Definitions	223
Initialization and Shutdown	225
IFXP_Initialize	226
IFXP_Shutdown	227
IFXP_Runtime_State_Changed	228
Display Configuration	229
IFXP_Display_Set_Mode	230
IFXP_Display_Get_Info	233
Canvas Management	234
IFXP_Canvas_Create	235
IFXP_Canvas_Destroy	237
IFXP_Canvas_Draw_Rect_Fill	238
Custom Image Format Unpacking	239
IFXP_Image_Open	240
IFXP_Image_Get_Data	242
IFXP_Image_Release_Data	243
IFXP_Image_Close	244
IFXP_Image_Check_Extension	245
Rendering	246
IFXP_Render_Start	247
IFXP_Render_Background_Start	248
IFXP_Render_Background	249
IFXP_Render_Background_End	251
IFXP_Render_Foreground_Start	252
IFXP_Render_Foreground_End	253
IFXP_Render_Display_Start	254
IFXP_Render_Display	255
IFXP_Render_Display_End	256
IFXP_Render_End	257
Native Fonts	258
IFXP_Text_Open_Font	259
IFXP_Text_Close_Font	261

Table of Contents

IFXP_Text_Get_Width	262
IFXP_Text_Get_Char_Height	263
IFXP_Text_Canvas_Create	264
IFXP_Text_Canvas_Destroy	266
Error Notes	267
IFXP_Display_Error_Note	268
FILE Functionality	269
IFXP_File_Open	270
IFXP_File_Read	271
IFXP_File_Seek	272
IFXP_File_Size	273
IFXP_File_Close	274
IFXP_File_Find_First	275
IFXP_File_Find_Next	277
IFXP_File_Find_Close	278
IFXP_File_Create	279
IFXP_File_Write	280
IFXP_Dir_Create	281
Timer Management	282
IFXP_Timer_Schedule	283
IFXP_Timer_Get_Current_Time	284
Benchmarking	285
IFXP_Benchmarking_Signal	286
Debug Support	287
IFXP_Error_Print	288
Memory Management	289
IFXP_Mem_Allocate	290
IFXP_Mem_Deallocate	292
Mutexes	293
IFXP_Mutex_Create	294
IFXP_Mutex_Destroy	295
IFXP_Mutex_Lock	296
IFXP_Mutex_Unlock	297
Other Methods	298
IFXP_Check_GL_Extension	299
Validating a New Porting Layer	300
 Chapter 11	
Included Third-party Software	301
FreeType	301
FreeType Features	301
Harfbuzz	304
Usage Notes	305
Version Shipped with Inflexion UI Runtime	305
Harfbuzz Licensing	305
NUbidi	305
Usage Notes	305
NUbidi Licensing	305

UCDN (Unicode Database and Normalization)	305
Usage Notes	306
Version Shipped with Inflexion UI Runtime	306
UCDN Licensing	306

Third-Party Information

Embedded Software and Hardware License Agreement

List of Figures

Figure 1-1. System Architecture Using Mentor Embedded Inflexion UI	15
Figure 1-2. Deployment Workflow	16
Figure 5-1. Call Sequence for Synchronous and Asynchronous Link Types	41
Figure 5-2. Module Function Call Sequence	42
Figure 5-3. Example Menu Hierarchy	44
Figure 5-4. Dynamic Menu Call Sequence	48
Figure 5-5. Direct Mode Plug-in Elements	51
Figure 6-1. The Flow of Rendering Calls into the Porting Layer	71

List of Tables

Table 4-1. Event Array Types	32
Table 5-1. Tokens	39
Table 5-2. Command Line Tool Switches	65
Table 6-1. General Porting Layer API Functions	68
Table 6-2. Memory Allocation Functions	69
Table 6-3. Native Fonts	73
Table 6-4. Mutex Support Function	73
Table 6-5. File System API	74
Table 9-1. General Type Definitions	116
Table 10-1. Type Definitions	223

Chapter 1

Introduction

This manual describes the Inflexion UI Runtime and demonstration projects, offers information on porting the product to different toolsets and platforms, and information on how to expose device functionality to the user interface (UI) theme writer.

In this chapter:

Mentor Embedded Inflexion UI Runtime	13
Included Components	13
Software Overview	14
Inflexion UI Runtime Architecture	14
Packages	17
Engine API Overview	17
Porting Layer Overview	18
Integration Layer Overview	18

Mentor Embedded Inflexion UI Runtime

The Inflexion UI Runtime (the on-device portion of the Inflexion UI system) is an XML-configurable UI layer that can be integrated with underlying application modules (providing native functionality to the deployed UI Theme), and it presents a graphically rich, animated interface, capable of supporting a wide variety of menu-based and instrumentation-based UIs.

The Inflexion UI Runtime is delivered as a set of C and C++ source files that compile into a library. It provides an “Engine” API that includes configuration functions, an “Integration Layer” that interfaces the engine to underlying native functionality and a “Porting Layer” reference implementation. The APIs to the Inflexion UI engine are C language and do not require the use of C++ for the modules or porting layer. In addition, the porting layer and integration layer present interfaces for use with Java modules and activities to simplify using Inflexion UI with Android activities. The product also includes a simple demonstration that shows basic module functionality.

Related Topics

[About Modules](#)

[Porting Overview](#)

Included Components

This release consists of the following parts:

- Inflexion UI engine product source code
- Default Inflexion UI Porting Layer implementation, based on the Nucleus OS kernel, Nucleus GRAFIX RS and Nucleus FILE products
- SimpleEdit Module to support basic text editing
- Command line “Integrator” tool for generating module integration C code and Java Interface files
- Demo application with pre-packaged sample theme

Related Topics

[About Modules](#)

Inflexion UI Express User’s Manual

Software Overview

The following topics detail the components comprising the Inflexion UI Runtime product set, and their interrelations with the host OS architecture:

- [“Inflexion UI Runtime Architecture”](#) on page 14
- [“Packages”](#) on page 17
- [“Engine API Overview”](#) on page 17
- [“Porting Layer Overview”](#) on page 18

Inflexion UI Runtime Architecture

This topic details the application layers comprising the Inflexion UI Runtime architecture, and its relationships with both the host operating system and the third-party modules/applications to which it has access at runtime.

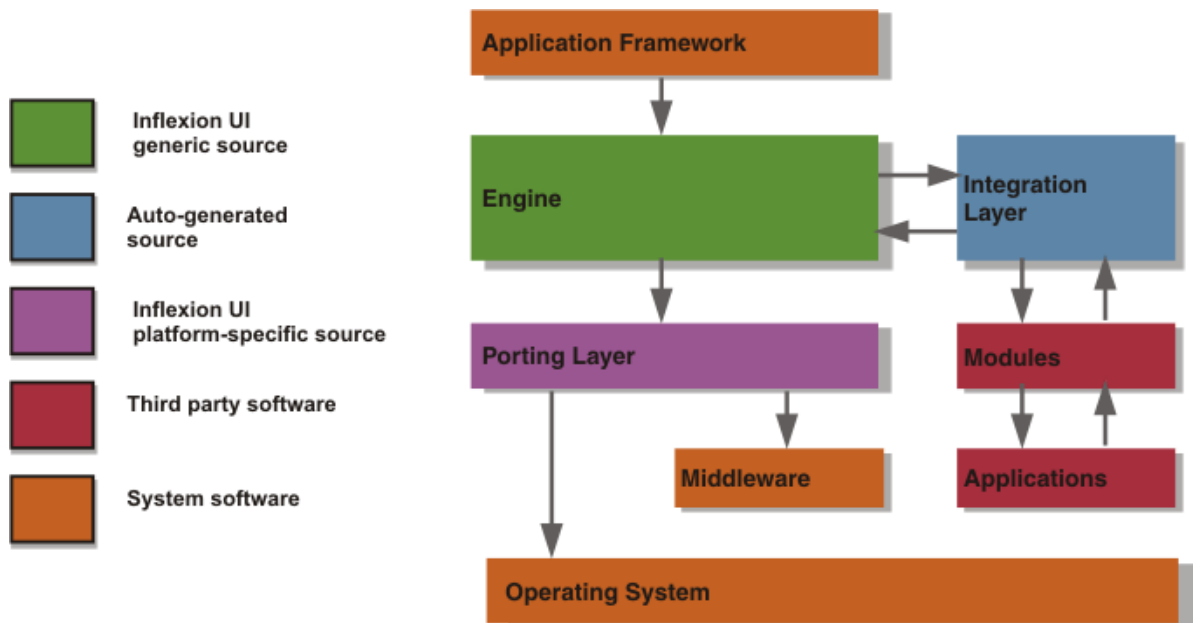
Figure 1-1. System Architecture Using Mentor Embedded Inflexion UI

Figure 1-1 shows the architecture of a system using the Inflexion UI Runtime component for rendering its user interface, including the porting layer. The porting layer might need to be re-implemented in part or in whole, depending on the target device platform and Inflexion UI Runtime configuration options.

The modules (and module definition files) that wrap native applications need to be implemented once per application in order to be portable to any device deploying the application. Once the modules are defined, the integration layer can be auto-generated using the Integrator tool supplied with the Inflexion UI Runtime.

The Application Framework in the diagram represents some kind of framework responsible for gathering user events, timer expiries, and application control signals and marshallng them onto a UI task thread. Then, if appropriate, they are passed into the Inflexion UI engine. The ReadyStart platform reference implementation comes with a sample framework integrated with the “event handler” component of the Nucleus OS middleware stack.

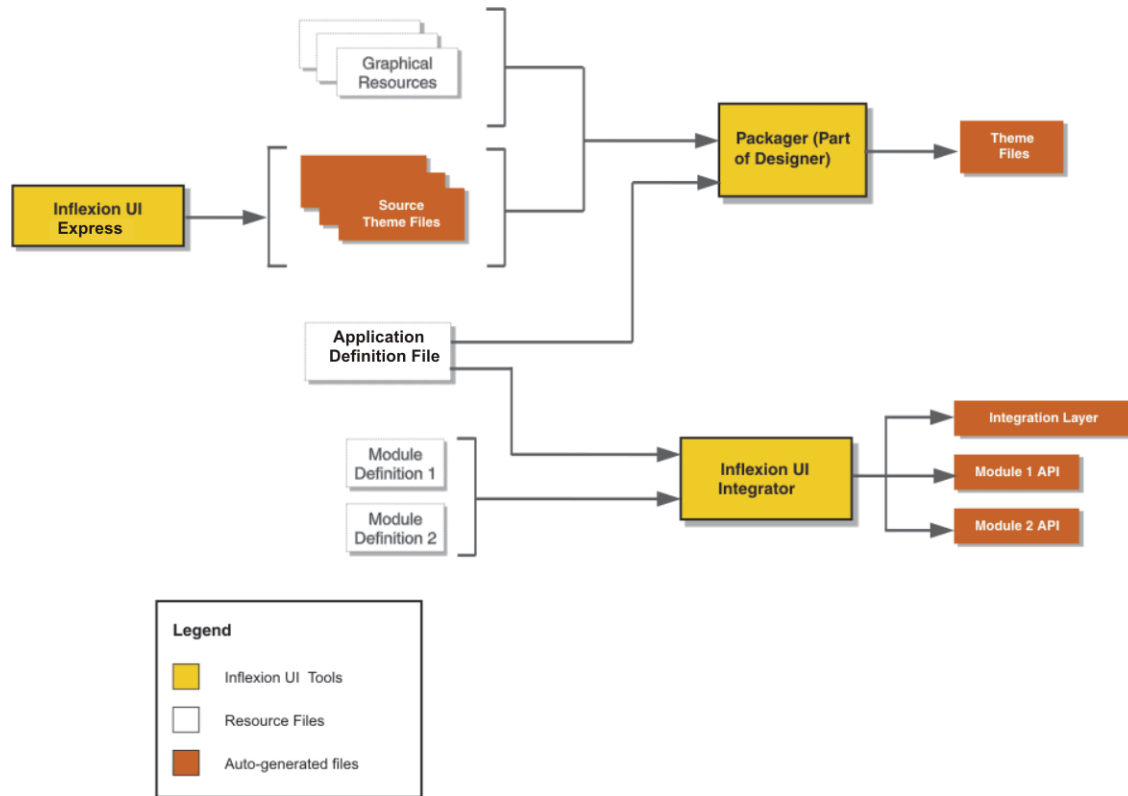
The Inflexion UI family includes the following two products:

- Inflexion UI Runtime
- Inflexion UI Express

Inflexion UI Express is a companion product to the Inflexion UI Runtime component. It is an application for Windows® PCs that enables the rapid creation, customization, and testing of new UI designs. Compelling, sophisticated designs can be constructed easily, without any programming or scripting, using simple (declarative) XML. When a UI design is ready, it can be packaged for deployment onto the device. Figure 1-2 illustrates the work flow, including

generating the integration layer and module headers for integrated modules using the UI Integrator tool (included in the engine release).

Figure 1-2. Deployment Workflow



The Inflexion UI Runtime component is an embedded software component that acts as an engine to display interfaces created with the Inflexion UI Express. It can be integrated onto any device with a graphical display. The engine reads the contents of the UI packages deployed on the device and uses them to render the UI onto the device screen.

The Inflexion UI Runtime component integrates with other software on the device via the following three APIs:

- **Engine API**

This is a set of functions implemented and exported by the Inflexion UI Runtime component. These are used by the host system to initialize, configure and control the UI, and pass it events such as key or touch panel events.

- **Integration Layer**

This is an automatically generated layer of C code that connects to existing application services and enables them to be accessed from within the UI design XML.

- **Porting Layer**

This is a platform-specific set of functions called by the Inflexion UI Runtime component and implemented to integrate the Runtime component onto a new platform. These manage canvasses, native fonts, and display canvasses on the screen, along with other platform-specific features such as file access. There is also a dependency on the underlying device kernel for some functionality, for example, timers, mutexes and dynamic memory allocation.

Typically, application services are provided by one or more “modules”, either implemented separately, or obtained as separate products. An “integration layer” of code manages interaction between the Inflexion UI Runtime component and multiple separate modules, as shown in [Figure 1-2](#), if modules are used. This integration layer is generated automatically using a command-line tool called the Integrator provided with the engine product.

The Inflexion UI Runtime component source bundle includes a module that can be used to add/edit functionality to text elements within a UI, and the demo application includes modules showing how a battery indicator may be included in the UI theme and updated with information from the device, how menus are populated and managed at run-time and how an application may be embedded within the Inflexion UI. The XML interface that uses the SimpleEdit Module is detailed in the *Mentor Embedded Inflexion UI Express User Guide*. The Runtime component also includes a utility library to create and manage the Universal Resource Indicators (URIs) used to affect actions within the Runtime component.

Packages

A typical UI configuration consists of a “package”. A package is a collection of XML files that specify pages, their layouts or other data, and graphical resources (for example, bitmaps and fonts). The Inflexion UI Runtime component uses the package contents to populate and present the UI. Integrated modules can provide additional data for the UI content and additional functions to be invoked from the user interface.

The package files are made available to the Inflexion UI Runtime component via a file system. The Runtime engine can be integrated with any file system, including Nucleus FILE, by way of the Porting Layer. The Inflexion UI Runtime component includes a simple ROM-based file system to allow UI Runtime resources and configurations to be compiled into the system image, in which case the porting layer file API is not used. In this mode, it is possible to continue integrating and testing the Inflexion UI deployment on a device even if the device file system is not yet available.

The demonstration application provided with the product includes a pre-generated C file containing file system data for the demo’s graphical resources and UI configuration. The original resource and configuration files are not included.

Engine API Overview

The core Inflexion UI engine is “event driven”, in that changes in UI appearance (including timed animations) rely on the external system for stimulus. Because there is no application

model imposed by choosing to use the Inflexion UI Runtime system, it is suitable for both task/thread-driven and event-driven systems.

The “normal” stimuli (timer event, user input, and so on) are applied using the “Engine” API. It is also possible for modules to effect changes in the UI via the Integration API.

The Engine API enables the host system to initialize and configure the Inflexion UI system, route user events to the engine, drive the time-based UI updates and shut down the Runtime component when it is not required.

Calls to the Engine API complete in real time, and the Inflexion UI engine is non-reentrant, so it is the duty of the application framework to serialize and marshal events (such as key presses and screen taps) before forwarding them on to the Runtime component via the Engine API. Note that the Inflexion UI Runtime processes events within a constant time, so that excessive UI activity results in a lower animation frame rate (with skipped frames) rather than a slower response time - this leaves the UI responsive at all times provided the application framework remains responsive to user actions under load. Access to the Engine API functions is protected by a mutex (provided the porting layer mutex implementation is complete). If the mutex implementation cannot suspend calling threads indefinitely, events must be marshalled onto a single execution thread before being passed into the Runtime component.

Porting Layer Overview

The Inflexion UI Runtime component uses a “Porting Layer” to manage canvasses, native fonts, and displays, along with other platform-specific functionality such as file storage access, and kernel features such as timers, mutexes and RAM allocation functions. This allows a custom accelerator or graphics engine to increase the overall system performance, and simplifies the integration of the Runtime component onto different devices.

The Inflexion UI engine calls the Porting Layer API during screen rendering operations to allow hardware acceleration functionality to be used. Details and specific use cases are addressed in the following sections.

Rendering the UI

The Inflexion UI engine renders the UI in a rectangular area on the screen using the Porting Layer. Other applications should not draw to the same area because this causes contention and disruption of the display. The exception to this is modules that serve directly rendered plugin elements. These can draw to the region reserved by the plugin element if the element is active. See the [About Modules](#) chapter for details.

Integration Layer Overview

The Inflexion UI Runtime component uses a generic integration layer interface, allowing one or more application modules to display information via the user interface. This open-ended

integration feature is provided by the Inflexion UI Integrator. It automatically generates the integration layer that links the engine and the modules. This enables integrated modules to be used in the following ways:

- provide menu and system data for display in the UI
- provide additional functions to be invoked through the UI
- provide components that do their own rendering to configured regions of the UI
- provide event handlers for user actions in the user interface

The invoked function is configured using URI syntax when an item, or button, is selected in the User Interface, for example, “*type://identifier?parameters*”. The Inflexion UI Runtime component understands two built-in link types: “*node://*” and “*theme://*”. Refer to the *Mentor Embedded Inflexion UI Express User’s Manual* for details on these.

The Integration Layer enables a module to handle additional link types on behalf of the Runtime component. When you select a menu item using a registered link type, the Inflexion UI engine calls the module via the Integration Layer to execute the required function.

A module can provide the Inflexion UI Runtime component with additional “menu” link types to open a menu component whose contents are populated dynamically (that is, at run-time) by the module. If this type of link is used in the UI, calls are placed to the module to provide data to populate that menu.

An example that includes both of these features is an MP3 player. An integrated MP3 player module provides a menu mode link to open a playlist. The playlist is shown as a menu in the Inflexion UI theme. Each item on the playlist can display artist, track name, and length. Selecting an item can cause the MP3 to play by invoking a function link implemented by the player’s module. The specific track to play is indicated by the body of the link, which is part of the data returned for each list item.

The UI can contain plug-in elements rendered by a module. Once the element is configured, the Inflexion UI Runtime component gives the module a rendering “context” that it can use to paint the element to the screen. An example of this is implementing a video player, so the module renders video content into a rectangular region of the UI, with that region fully configured in XML. The “event handler” element is a special case of the “plugin element” feature because it has no screen rendering capabilities. It allows the module to perform actions in response to user interaction with appropriately configured screen elements.

API References

Mentor Embedded Inflexion UI does not require any programming in order to display basic UI themes created using the Inflexion UI Express tool, but it does require integration with the host device. This manual details the various APIs defined by the Inflexion UI Runtime, why they are

needed, and how they are used. There are two aspects to the APIs: the “Application API” and the “Integration APIs”.

- **Application API**

Inflexion UI must be configured, started, and “stimulated” following user action. The functions to do this are grouped into the “engine” API, prefixed by “IFXE_”.

- **Integration APIs**

At a bare minimum, Inflexion UI relies on a set of typical kernel features (such as timers, mutexes, and RAM allocation functions for example) and an underlying graphical rendering layer to display images to the device screen. Typically, Inflexion UI Runtime also makes use of some device-based file access system and possibly a device-based font/font engine. This set of dependencies is encapsulated in a set of methods called from the Inflexion UI Runtime code, called the “Porting Layer”. Each porting layer function has an “IFXP_” prefix.

Inflexion UI allows an arbitrary set of device capabilities to be exposed to the UI theme through the Inflexion UI engine using the concept of “modules”. Each module defines the capabilities that can be used in a theme in a module definition file, which is then used to auto-generate an “integration layer” that connects the Inflexion UI engine component to the native capabilities of the device. This process generates prototypes for a set of methods that each module must implement (the “Module Integration” API, with function prefix name “IFXM_”), and Inflexion UI Runtime offers a set of methods (the “Engine Integration” API, with the function prefix name “IFXI_”) for modules to make requests of the Inflexion UI Runtime and an additional set of methods called the “URI Parser” API (with the function prefix “IFXU_”) to help module writers work with the special form of URI link strings used by the Inflexion UI engine component.

Related Topics

[Engine API Reference](#)

[Porting API Reference](#)

[Module Integration API Reference](#)

[Included Third-party Software](#)

Chapter 2

Getting Started

This chapter explains how to build the Inflexion UI engine library and provides a demonstration that illustrates the basic usage of the Inflexion UI engine. These sections explain how to execute the demonstration on target hardware and verify its functionality.

In this chapter:

Initializing Mentor Embedded Inflexion UI Runtime Component.....	21
Building the Mentor Embedded Inflexion UI Runtime Package	21
Including the Mentor Embedded Inflexion UI SimpleEdit Integrated Module Component	22
Including the Mentor Embedded Inflexion UI Porting Layer Package	22
Including the Freetype Package	22

Initializing Mentor Embedded Inflexion UI Runtime Component

The Inflexion UI Runtime component is initialized as part of the *ReadyStart for Nucleus* initialization process. By default, its initialization run level is set to “15” in the file *nucleus/os/ui/inflexionui/framework/.metadata*.

Building the Mentor Embedded Inflexion UI Runtime Package

The Inflexion UI Runtime package is built as part of the ReadyStart build system if it is enabled. Because the Inflexion UI Runtime package is enabled by default, the configuration file being used must explicitly turn off support to remove Inflexion UI Runtime from the ReadyStart system (that is, add “nu.os.ui.ifx.port.enable=0” to the configuration file). The Inflexion UI Runtime package depends directly on GfxRS (nu.os.ui.grafxrs) and the display driver (nu.os.drvr.display) for rendering services, Input Management (nu.os.ui.input_mgmt) for marshalling key/touchpanel events and Nucleus FILE (nu.os.stor.file) for platform file storage. Each of these subcomponents will have platform-specific dependencies; refer to the BSP documentation for more information.

Including the Mentor Embedded Inflexion UI SimpleEdit Integrated Module Component

The “SimpleEdit” component is an Inflexion UI module offering limited text edit functionality to Inflexion UI interfaces. Although it is included in the ReadyStart build system by default, you can disable it by adding “nu.os.ui.ifx.mod.simpleedit.enable=0” to the configuration file.

Including the Mentor Embedded Inflexion UI Porting Layer Package

The Inflexion UI Porting layer supports the common Inflexion UI Engine package, providing platform-specific services. It is required if the engine package is enabled, and is enabled by default.

Including the Freetype Package

The Freetype package provides font support to the Inflexion UI engine package. It is built as part of the ReadyStart build system, and is enabled by default. It can be disabled by adding “nu.os.ui.ifx.freetype.enable=0” to a config file.

Chapter 3 Configuration

This chapter details the options available to configure Inflexion UI Runtime for a target device, all of which are specified in the ReadyStart configuration file. Simply edit the configuration settings in the file to fit the device you are using.

In this chapter:

File System Support	23
Rendering Mode	24
Canvas Mode	25
Canvas Order	26
Open GL Render-mode-specific Options	26
Performance	27
Benchmarking Support	27
Feature Support	28
Development Support	29
Porting Layer Configuration	29

File System Support

use_rom_files

Theme resources are compiled into the executable image and stored in a global array `g_romFileTable`. The theme packager application provided with the Inflexion UI Express tool is required to package a theme in this form. For example:

```
nu.os.ui.ifx.eng.use_rom_files=true
```

This option is enabled by default. It can be disabled to reduce the size of the native library.

use_platform_files

Theme resources are read from a device file system, via an API in the porting layer.

The packager application provided with Inflexion UI Express packages themes into a folder structure ready for deployment onto the device file store. The Runtime component reads the resources from the file store when it is started. The drive(s) searched are defined by:

- `package_drives` (for example, “C,B,Z” searches these drives in order). The Porting Layer performs the drive letter’s mapping to the platform-specific locations.

For example:

```
nu.os.ui.ifx.eng.use_platform_files=true  
nu.os.ui.ifx.eng.package_drives="B,Z"
```

Rendering Mode

One render mode may be selected for use; the other render modes must be set to “false”. The following render modes are supported in this release:

render_internal

The Inflexion UI engine contains a built-in “2.5D” drawing mode. Although it performs most of the same functionality of the other 3D modes without using an OpenGL implementation, it cannot draw 3D meshes. This is the default mode. For example:

```
nu.os.ui.ifx.eng.render_internal=true
```

render_internal_compressed

In this mode, the Inflexion UI engine manipulates graphical data in compressed form to conserve RAM usage, but the graphics can only be drawn in 2D. The compressed format improves the RAM footprint by 20-30%, at the expense of reducing animation frame rates by 20-30%. This option is disabled by default. For example:

```
nu.os.ui.ifx.eng.render_internal_compressed=true
```

render_direct_opengl

The Inflexion UI engine uses OpenGL in direct mode to generate frames. The OpenGL implementation draws the whole frame to display every frame. It is best suited to hardware-accelerated OpenGL implementations. The Board Support Package must include an OpenGL 1.1 driver implementation for this mode to be used. This option is disabled by default.

```
nu.os.ui.ifx.eng.render_direct_opengl=true
```

render_direct_opengl_20

The Inflexion UI engine uses OpenGL ES 2.0 in direct mode to generate frames. The OpenGL implementation draws the whole frame to display every frame. It is best suited to hardware-accelerated OpenGL implementations. The Board Support Package must include an OpenGL 2.x driver implementation for this mode to be used. This option is disabled by default.


```
nu.os.ui.ifx.eng.render_direct_opengl_20=true
```

Canvas Mode

The following modes determine what format the Inflexion UI Runtime component canvas uses. In general, the canvas format must match the display format. However, the Porting Layer can be written to perform conversion between different formats for improved efficiency. Valid options are:

canvas_mode_8888

This setting forces the Inflexion UI Runtime component to use the RGBA8888 format (32 bits per pixel). This mode is disabled by default. For example:

```
nu.os.ui.ifx.eng.canvas_mode_8888=true
```

canvas_mode_888

This setting forces the Inflexion UI Runtime component to use the RGB888 format (24 bits per pixel). This mode stores an alpha value and color values for each pixel, and it is intended for use with systems that overlay a background in hardware. This mode is disabled by default. For example:

```
nu.os.ui.ifx.eng.canvas_mode_888=true
```

canvas_mode_565

This setting forces the Inflexion UI Runtime component to use the RGB565 (16 bits per pixel) format. This mode is disabled by default. For example:

```
nu.os.ui.ifx.eng.canvas_mode_565=true
```

canvas_mode_1555

This setting forces the Inflexion UI Runtime component to use the RGB1555 (16 bits per pixel) format. This mode is disabled by default. For example:

```
nu.os.ui.ifx.eng.canvas_mode_1555=true
```

canvas_mode_444

This setting forces the Inflexion UI Runtime component to use the RGB444 (12 bits per pixel) format. This mode is disabled by default. For example:

```
nu.os.ui.ifx.eng.canvas_mode_444=true
```

Canvas Order

The following settings are used to specify the color channel order for the channel being used:

canvas_order_rgb

This setting tells the Inflexion UI engine that the canvas being used has channel order “Red, Green, Blue”. This is the default setting.

The canvas_order_rgb setting sets the canvas channel order to be “Red, Green, Blue, (Alpha)”. If disabled, the channel order is assumed to be “Blue, Green, Red, (Alpha)”. This mode is enabled by default. For example:

```
nu.os.ui.ifx.eng.canvas_order_rgb=true
```

Open GL Render-mode-specific Options

The following settings only apply if the render_direct_opengl or render_direct_opengl_20 render modes are used.

bpp

This selects the bits per pixel of the device, and will be used during the selection of the “best fit” EGL configuration. The default value is 32.

```
nu.os.ui.ifx.eng.bpp=32
```

depth_buffer_size

Sets the size of the Open GL depth buffer in bits. The default value is 24.

```
nu.os.ui.ifx.eng.depth_buffer_size=24
```

swap_interval

Set the swap interval value for synchronization of the buffer swap operation. A value of 0 will allow the buffers to swap as soon as they’re ready, but the output will be prone to “tearing” issues. A value of 1 or more will ensure that the front buffer has been drawn to screen before the front and back buffers may be swapped. The default value is 1.

```
nu.os.ui.ifx.eng.swap_interval=1
```

msaa_samples

Select the level of antialiasing (or its equivalent) to use. This value will be used when selecting the best fit EGL configuration. The default value is 0 (no antialiasing); values greater than 0 (usually powers of 2) will “smooth” pixelation “jaggies” at the expense of a potential drop in frame rate when animating.

```
nu.os.ui.ifx.eng.msaa_samples=0
```

Performance

use_big_endian

This setting should be enabled on platforms that store multi-byte data in big endian format. It is disabled by default. For example:

```
nu.os.ui.ifx.eng.use_big_endian=true
```

Benchmarking Support

enable_bm_frame_rate

If this define is included, the Inflexion UI Runtime component provides the porting layer with the information required to calculate average frame rates and page opening times. It allows the demonstration to display information in frames per second for frame rates, and milliseconds for page opening times. This define is disabled by default. For example:

```
nu.os.ui.ifx.eng.enable_bm_frame_rate=true
```

enable_bm_heap_usage

If this define is included, the Inflexion UI engine tracks the current “local” maximum (over the last animation period) and global maximum “heap” RAM usage and allows the demonstration to display information in kilobytes. If the `NU_STACK_CHECKING` define is enabled in *plus_cfg.h*, this mode also monitors the UI Runtime task stack usage and displays the information on screen. This define is disabled by default. For example:

```
nu.os.ui.ifx.eng.enable_bm_heap_usage=true
```

Feature Support

use_stylus

This define enables “pointer” support in the Inflexion UI Runtime component (for stylus taps, drag events and so forth). It is enabled by default. For example:

```
nu.os.ui.ifx.eng.use_stylus=true
```

use_mouseover

For devices that support mouse-like pointing devices, you may enable or disable support for “mouseover” displacements within the runtime. When enabled, the theme writer may animate UI elements when the mouse pointer is hovering over the element. This is enabled by default.

For example:

```
nu.os.ui.ifx.eng.use_mouseover=true
```

use_bitmapped_fonts

This define enables the use of bitmapped fonts (fonts with up to 144 characters pre-rendered into a bitmap). Bitmapped fonts are created using the Inflexion UI Express. They are much quicker to render from than native fonts and are limited to Western European languages. This is enabled by default. For example:

```
nu.os.ui.ifx.eng.use_bitmapped_fonts=true
```

use_native_fonts

This allows font rendering using an external font engine. the Inflexion UI Runtime component contains a reference implementation of this feature using the FreeType open source font engine, via the relevant porting layer API. This is enabled by default. For example:

```
nu.os.ui.ifx.eng.use_native_fonts=true
```

use_plugin_elements

This define enables use of plug-in elements offered by modules (see the [About Modules](#) chapter). This is enabled by default. For example:

```
nu.os.ui.ifx.eng.use_plugin_elements=true
```

validate_projects

This define enables checksum verification on loaded themes. If themes are modified from their original packaging, any attempt to load them fail. If the GUID embedded into the theme does not match the GUID used by the auto-generated integration layer, validation fails. This ensures that the theme is always displayed on a suitable device. This is disabled by default due to the performance cost, but may be enabled in production devices for security purposes as needed. For example:

```
nu.os.ui.ifx.eng.validate_projects=true
```

use_deferred_component_loading

This setting defers the loading of component template files until a component instance using this file is placed in the currently active layout. This may speed up load times for themes where many components are used on a single node template, but some are only in layouts not initially used. The default setting is “true”. For example:

```
nu.os.ui.ifx.eng.use_deferred_component_loading=true
```

Development Support

debug_mode

This mode will enable some additional diagnostic checks and logging within the runtime, along with some additional memory management. There is a performance cost, so this is disabled by default. For example:

```
nu.os.ui.ifx.eng.debug_mode=true
```

Porting Layer Configuration

The porting layer (the thin layer between the core Inflexion UI runtime and the Nucleus operating system/drivers) has a small number of configuration settings.

a_drive_map

When using the use_platform_files mode, the “A” drive used inside the Inflexion runtime should be mapped to a Nucleus FILE drive letter. The default mapping is to “A:”.

b_drive_map

When using the use_platform_files mode, the “B” drive used inside the Inflexion runtime should be mapped to a Nucleus FILE drive letter. The default mapping is to “B:”.

c_drive_map

When using the `use_platform_files` mode, the “C” drive used inside the Inflexion runtime should be mapped to a Nucleus FILE drive letter. The default mapping is to “C:”.

default_font_header_file_name

When using the `use_native_fonts` setting, you may specify a default TTF file that has been placed in binary form in a header file. If a font is requested but not found, the default font will be used instead. The default font header file in the release is *inc/font_mplus_1p.h*.

Chapter 4

Evaluating Performance and Benchmarking

The demonstration supports running in a benchmarking mode, which answers a commonly asked question: “What kind of performance can I expect for X MIPS?”. The answer is particularly difficult in the case of the Inflexion UI engine. At rest, the engine has virtually no impact on performance. During animation, it can potentially use 100% of the available CPU cycles. In addition, two different UI themes can have very different impacts on frame drawing times during animation.

Benchmarking mode allows a given theme to be tested for average frame rate (in frames per second, or “fps”), measure menu opening times, and monitor the heap and stack usage of the Inflexion UI engine.

In this chapter:

Configuring the Engine	31
Configuring the Porting Layer and the Demonstration	31
Configuring the Averaging of the Frame Times.	33
Outputting the Frame Rate Readings	33
Benchmarking and Testing Guidelines	34
Benchmarking Tips	34

Configuring the Engine

The Inflexion UI engine must be configured to emit the signals required by the demonstration in benchmarking mode. *nucleus* This is done by enabling one of the options in the ReadyStart configuration file. If you want to see the average frame rate, enable the *enable_benchmarking_frame_rate* option. If you want to see the Heap RAM usage, enable the *enable_benchmarking_heap_usage* option.

Configuring the Porting Layer and the Demonstration

The demonstration and the porting layer automatically detect that benchmarking is enabled in the Inflexion UI engine. No configuration is required for benchmarking to function. There are a number of parameters in the benchmarking process that can be modified. These are explained in this section.

The demonstration and the porting layer load the theme (either compiled into the demo image or from the file system), issue an endless sequence of “down” events, and either display the current frame rate in the screen’s bottom-left corner, along with a measurement of time taken to open a page, (that is 0 in the default key sequence case), or figures for current heap usage, local maximum heap usage (max heap usage since the last “animation complete” point), global maximum heap usage and the Inflexion UI task maximum stack usage (if NU_STACK_CHECKING is enabled in Nucleus PLUS).

Configuring the Automated Key Sequence

You are free to configure any sequence of keypresses and pauses required. You can also disable the automated user input feature and navigate freely while benchmarking.

Each event takes the form of “event, parameter 1, parameter 2”. “event” can be a keypress, stylus tap, loop, or pause.

Table 4-1. Event Array Types

Event	Description
Spoof_Key_Press	parameter 1 should be the relevant the Inflexion UI Runtime component IFX_Key action.
Spoof_Stylus_Tap	parameter 1 and parameter 2 are the x and y coordinates of the stylus tap relative to the screen’s top-left-hand corner (y increases in the downwards direction).
Spoof_Event_Pause	parameter 1 is the number of milliseconds to pause (allowing, for example, a static animation to run).
Spoof_Event_Loop	The event at the sequence’s start is executed, and the sequence continues.
Spoof_Touch_Down	parameter 1 and parameter 2 are the x and y coordinates of the stylus tap relative to the screen’s top-left-hand corner (y increases in the downwards direction).
Spoof_Touch_Move	parameter 1 and parameter 2 are the x and y coordinates of the stylus tap relative to the screen’s top-left-hand corner (y increases in the downwards direction).
Spoof_Touch_Up	parameter 1 and parameter 2 are the x and y coordinates of the stylus tap relative to the screen’s top-left-hand corner (y increases in the downwards direction).

For example, the following array navigates down twice, selects the current item, waits for one second, returns to the original menu, then navigates up twice, and repeats the process indefinitely:

```
static SPOOF_INPUT_EVENT IFXD_Events[] = {  
    {Spoof_Key_Press, KDown, 0},  
    {Spoof_Key_Press, KDown, 0},
```



```
{Spoof_Key_Press, KSelect, 0},
{Spoof_Event_Pause, 1000, 0},
{Spoof_Key_Press, KBack, 0},
{Spoof_Key_Press, KUp, 0},
{Spoof_Key_Press, KUp, 0},
{Spoof_Event_Loop, 0, 0}
};
```

The theme should be fully loaded prior to the sequence's start to allow the automated sequence code to function properly. The application then sends each element in the event array to the Inflexion UI framework event queue, shown in the example:

```
UNSIGNED engine_msg[IFXF_QUEUE_ELEMENT_SIZE];
/* Setup and send the event */
engine_msg[0] = IFXFTouchDown;
engine_msg[1] = event.param1;
engine_msg[2] = event.param2;
NU_Send_To_Queue(&IFXF_Queue, engine_msg, IFXF_QUEUE_ELEMENT_SIZE,
NU_NO_SUSPEND);
```

Configuring the Averaging of the Frame Times

The porting layer monitors frame updates over a certain period (the default is five seconds), then calculates the average frame rate over that period. This period can be adjusted—shortening it will make the frame rate more “volatile” but can reveal parts of the UI that have significant performance issues. Adjust the value of IFXP_BENCHMARK_FRAME_COUNT_TIME in the file *nucleus/os/ui/inflexion/porting/inc/ifxui_porting_nucleus.h* to set the required period in milliseconds.

Outputting the Frame Rate Readings

The current frame rate displays using Nucleus Serial Output by default. Simply attach a serial cable to the hardware and connect via a terminal (or, in the case of QEMU, press CTRL-ALT and 3). Alternatively, you can display the benchmarking results on screen via Nucleus GfxRS by uncommenting RS_OUTPUT_METHOD in the file *nucleus/os/ui/inflexionui/porting/src/ifxui_porting_benchmarking.c*. The output's location can be modified by selecting the appropriate define in the *nucleus/os/ui/inflexion/porting/src/ifxui_porting_benchmarking.c* file. For example, uncommenting BENCHMARK_TOP_LEFT moves the output to the display's top left-hand corner. This allows the frame rate output region to be set to a location not frequently updated by the Inflexion UI Runtime component to make reading the frame rate easier.

It is straightforward to add further benchmark results transfer mechanisms. The following functions must be implemented using the appropriate mechanism to add another means to output benchmarking data:

- extern VOID IFXD_Initialize_Output(VOID);
- extern VOID IFXD_Prepare_Output(VOID);

- extern VOID IFXD_Output_Frame_Rate(unsigned int frameRate);
- extern VOID IFXD_Output_Menu_Open_Time(unsigned int menuOpenTime);
- extern VOID IFXD_Output_Mem_Usage(unsigned int currentHeap, unsigned int localHeap, unsigned int globalHeap, unsigned int stack);

Benchmarking and Testing Guidelines

This topic offers some suggestions and hints regarding benchmarking and gives an example test framework to test a theme.

Benchmarking Tips

- Ensure that the theme has no “static” animations enabled when measuring frame rate using the auto navigate feature, because they disrupt the auto navigation.
- Many themes can be animated at a greater rate than the 33 frame per second default. Therefore, the value of IFX_FRAME_INTERVAL should be changed to 20 or even 10 ms if the reported frame rate at a 30ms interval is 30 or higher.
- The automated key sequence should not involve opening or closing menus for accurate frame rate figures.
- When benchmarking page opening times, the same page should be opened multiple times and an average time calculated because the page opening time can vary.
- A large portion of the time required to open a page is a result of loading and parsing the template file. Template files are cached, therefore if the page template or any component templates are used by an open page, that page opening time is much less than if the template is loaded from scratch.
- Heap usage usually reaches a maximum value in the deepest menu level of a theme. Any automated or manual testing should ensure that all menus are opened during the course of testing.
- The Inflexion UI engine is designed to be robust in the case of low-memory conditions. When the Heap is exhausted during the normal operation of the engine, an error note displays informing you that the memory is full. Theme operation continues from the same point if possible. Otherwise, it continues from the top-level menu.
- Maximum stack usage can occur anywhere in a theme and requires an exhaustive functional test of the theme to ensure that every UI flow option is selected.

Testing Framework

The following test examples are taken from internal test plans. They are examples of the types of tests that should be performed on your themes to ensure end-user satisfaction.

General Layout Tests

- For each layout, on each page, the following should apply to all icons unless defined as such in the layout:
 - They should be visible
 - They should not overlap
 - They should not be clipped by screen edges
 - They should be free of scaling artifacts
- For each layout, on each page, the following should apply to all captions unless defined as such in the layout:
 - They should be visible
 - They should be legible
 - They should not overlap
 - They should not be clipped, truncated, or badly wrapped
 - They should be free of scaling artifacts

Page Navigation

- Any defined keys or stylus taps must switch to the expected new layout when triggered for each state on each page.
- The animation used to show each change should be quick, smooth, and aesthetically acceptable.
- All menu component items must be selectable.

Item Actions

Invoking the selected item results in the appropriate action for each layout on each page. For example, menu component items should open the correct pages, assuming the menu items are UI flow-related, and application items should launch the correct application.

Other Animations

The initial theme animates smoothly if it is configured to do so on the Inflexion UI Runtime component engine startup.

Robustness

- The application must be robust for all animations/keys. The theme must respond correctly to any sequence of rapid key/stylus presses, especially when the second (and subsequent) presses occur during an animation started by the first press.

- The theme should respond correctly to menu options selected quickly during animation. For example, the theme changer or application exit.

Chapter 5

About Modules

Modules allow the core Inflexion UI functionality to be extended in a generic manner to provide richer UI functionality than that achieved using pre-defined menus and images. Services that a module can offer include expanding “field” tokens, populating “dynamic” menus, rendering images to a portion of the screen, and performing actions when you select a particular type of link.

In this chapter:

Module Creation	38
Module Use	38
Advice for Module Writers	39
Modules and IFXM Functions	39
IFXM Function Formats	39
Modules Supporting Links	40
Modules Supporting Field Tokens	41
Modules Supporting Dynamic Menus	43
Modules Supporting Integrated Applications	48
Module Definition File	51
Fields	52
Links	53
Menus	54
Elements	54
Triggers	55
SimpleEdit Module Example	55
Device Definition File	56
Portable Key Codes	56
Format	57
Graphical Capabilities	57
Languages	58
Display Mode	58
Design Sizes	58
Device Definition File Example	60
Application Definition File	62
Format	62
supportedModules	63
entryPoints	63
Application Definition File Example	64

Mentor Embedded Inflexion UI Integrator.....	64
Usage.....	64
Integrator Example	65

Module Creation

The process for creating a module that integrates with the Inflexion UI Runtime component is as follows:

1. Decide what the module offers the theme designer

This means defining the links, field tokens, menus, and plugin elements that the module supports and any relevant parameters to these.
2. With this information, construct a module definition file, including a unique module name.

A module definition file is an XML file that specifies the module in a manner understood by the Integrator tool.
3. Use the Integrator tool to generate the module header file.
4. Implement all functions defined in the module header and any other functionality required to support these functions.



Note

Modules can be implemented in the C programming language or in Java. Because the required language is specified on a per-application basis, the same module definition can be used on different devices, with modules implemented in differing languages. For the Nucleus OS port of Inflexion UI Runtime, the expectation is that all modules will be written in C.

Module Use

When a module name is defined, it can be used in an Inflexion UI application by adding a reference to the module definition file in the application definition file associated with the Inflexion UI application. When the Integrator tool is executed, an Integration Layer is generated that interfaces the Inflexion UI Runtime component with all of the included modules. When compiled into the application, this Integration Layer calls the appropriate module API function at the relevant time.

An application using the Integrator-generated Integration Layer must be linked against all relevant module libraries.

Advice for Module Writers

The following should be considered when designing and writing a new module:

- Portability

The Inflexion UI Runtime component offers module writers a wrapped subset of the C language runtime library (RTL). Module writers who confine their use of the RTL to the functions located in the *nucleus/os/include/ui/ifxui_rtl.h* header file benefit from improved ease of porting between compilers.

- Module-specific Error Codes

The general set of the Inflexion UI Runtime component error codes are defined in the IFX_RETURN_STATUS enum tag located in the *nucleus/os/include/ui/ifxui_defs.h* header file. If a module defines error codes specific to that module, then they must be positive integers, greater than or equal to the value of IFX_ERROR_USER_DEFINED.

Modules and IFXM Functions

Modules generally perform one or more of the following roles for the Inflexion UI Runtime component:

- Handle special link actions
- Provide the current value of a “dynamic” (for example, “field”) token and notify the Inflexion UI Runtime component if it changes
- Provide the information to display a “dynamic” menu on request
- Offer the Inflexion UI Runtime component an embeddable entity that renders content to a portion of the screen
- Perform an action in response to a specified user input event

The IFXM functions implemented by a module depend on the contents of that module’s definition file.

IFXM Function Formats

The [Module Integration API Reference](#) chapter discusses the IFXM functions required by the Inflexion UI Runtime component. It also provides a template of the function using the following tokens retrieved from the module definition file:

Table 5-1. Tokens

Token	Description
<module name>	The module name.

Table 5-1. Tokens (cont.)

Token	Description
<link prefix>	The prefix of the link URI used to create the menu or element. See the linkPrefix attribute in the module definition.
<field name>	The name of the field the function is servicing.
<link params>	A list of parameters generated based on the link URI specified in the module definition.
<scan code>	The scan code as a string, or one of the pre-defined key names.



Caution

Some C compilers limit function names to a maximum of thirty-one characters. This does not allow the naming scheme defined in this section to be used, therefore you must rename the functions for these compilers.

Modules Supporting Links

Specific function declarations are generated for each different link type supported by a module. The parameter listings for link IFXM methods are also generated based on the link elements defined in the module's definition. For example, a module designed to handle a link of the form:

```
mp3Play://c:/music/test.mp3?loop=true;duration=10
```

would have the following in the module definition file:

```
<link linkPrefix="mp3Play"
linkType="functionSync" bodyType="string">
  <param name="loop" type="string" />
  <param name="duration" type="int" />
</link>
```

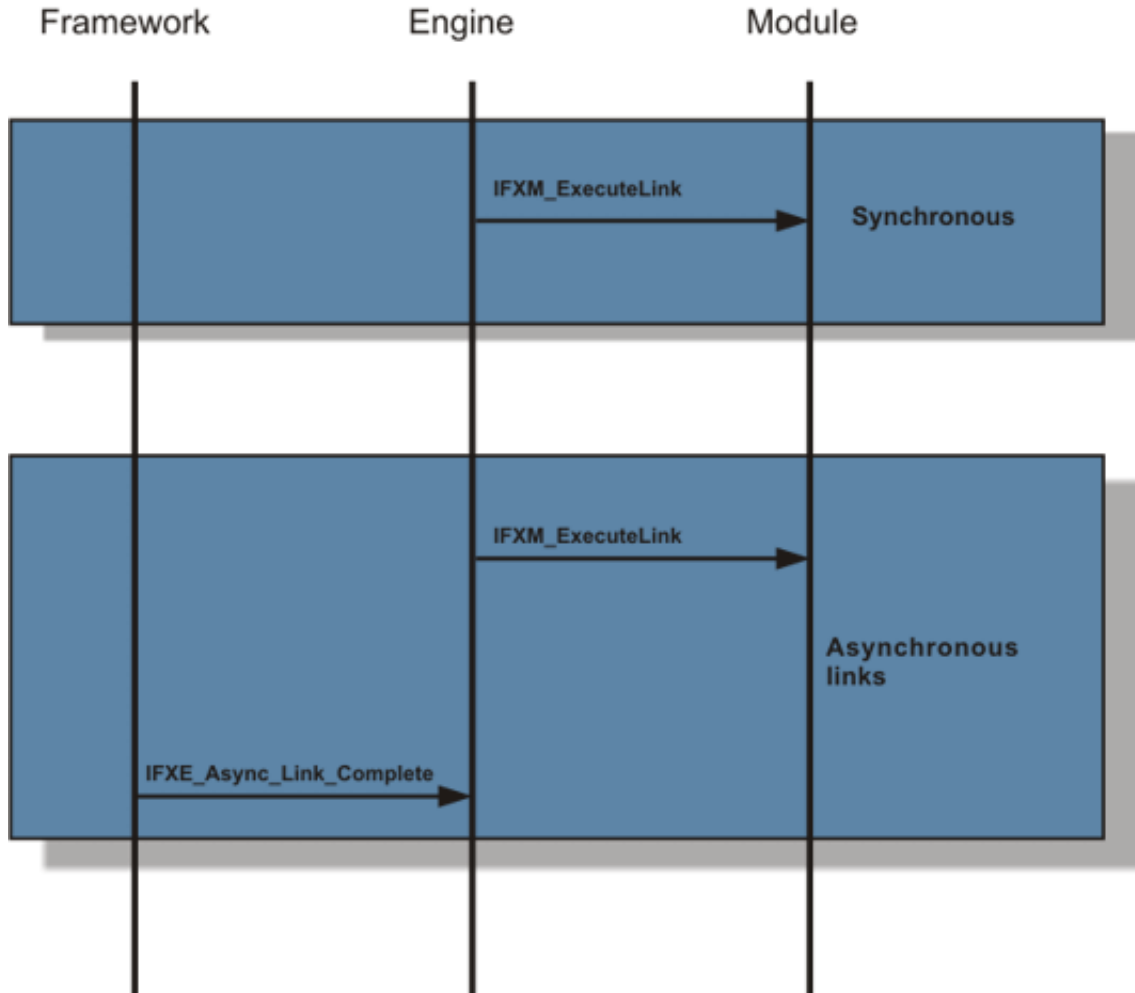
which translates to the following function declaration:

```
IFXM_MediaPlayer_LaunchLink_Mp3Play(
    IFX_HMODULE hModule,
    IFX_HMENU* phMenu,
    IFX_WCHAR* szBody,
    IFX_WCHAR* loop,
    INT32 duration);
```

The mediaPlayer module must implement this function, which is called from the auto-generated Integration Layer module.

Figure 5-1 shows the call sequence for synchronous and asynchronous link types. Note that in the asynchronous links, the engine must be brought out of the “asynchronous link suspend” state via a call to IFXE_Async_Link_Complete.

Figure 5-1. Call Sequence for Synchronous and Asynchronous Link Types



Modules Supporting Field Tokens

Field data returned by modules is raw data, which comes in the following four different types:

- int - Integer data. Within the module implementation this is of type IFX_INT32.
- float - Floating point data. Within the module implementation this is of type “float”.
- time - Time/date data. Within the module implementation this is of type IFX_TIME.
- string - Text data. Within the module implementation this is represented by arrays of IFX_WCHAR values.

- boolean - True/False data. Within the module implementation this is of type IFX_BOOL.

For each string-type field, a separate GetFieldStringSize function is required, used to establish how much space the Inflexion UI Runtime component requires to store the value of the string field. One single, common, GetFieldData function is required to return the actual string, with a request handle set in the GetFieldStringSize call to resolve the disambiguation. The “int”, “float”, “boolean”, and “time” type raw data types only require GetFieldData.

For example, a module supporting a global (not tied to a particular menu or menu item) field token playCount recorded as an integer within the module contains the following in the module definition file:

```
<field name="playCount" dataType="int" defaultValue="%d" />
```

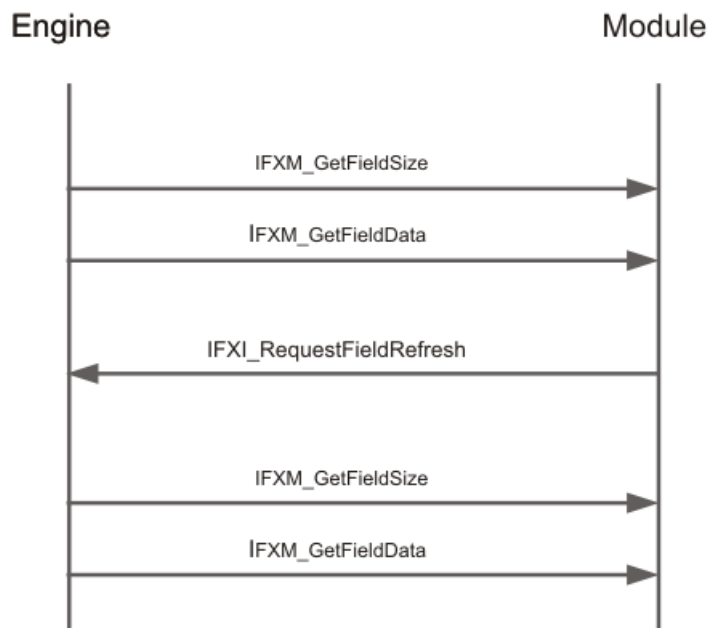
which translates to the following function declaration:

```
IFXM_MediaPlayer_GetFieldIntData_PlayCount(  
    IFX_HMODULE hModule,  
    IFX_INT32* pData);
```

The module must implement this function, which is called from the auto-generated Integration Layer.

Figure 5-2 indicates the pattern of function calls used when a module supports field data. The first interaction shows the engine evaluating the field token when it’s first used in the UI theme. At some later point, the module notifies the engine that the value of the field has changed, and the engine subsequently re-evaluates the field token value.

Figure 5-2. Module Function Call Sequence



Modules Supporting Dynamic Menus

Dynamic menus are ones where the contents of the menu are determined at runtime. To allow a theme designer to create the menu template used to display the content of the dynamic menu (something that obviously cannot be done at runtime) the module must specify what data is available for a particular menu. In the Inflexion UI system, each datum is encapsulated in a “field”.



Note

It is perfectly valid to have menu elements that define no fields. Essentially, they define a menu name that the theme designer can associate with a particular “view” on an identical data set.

The last point to note about the menu structure defined by the module definition file is how the menu opening link is related to the menu structure and the implications for implementing the menu functionality in the module. The menu open link (defined in the links section of the module definition file) references a single node on the menu hierarchy tree. This means the following:

- The menu name returned to the Runtime component is the menu name of the node the link is associated with, or the name of any of the descendant nodes.
- The name returned cannot be the name of any of the ancestor nodes.
- The set of field retrieval functions that must be implemented for that link encompass all the fields from the ancestor nodes, the field in the node associated with the link and all the fields from all the descendant nodes.

The module implementer does not have to worry about copying the menu names into the module. The Integrator tool generates an enumerated list of permitted menu names, and translates the enum value to a properly constructed string that returns to the engine in the integration layer.

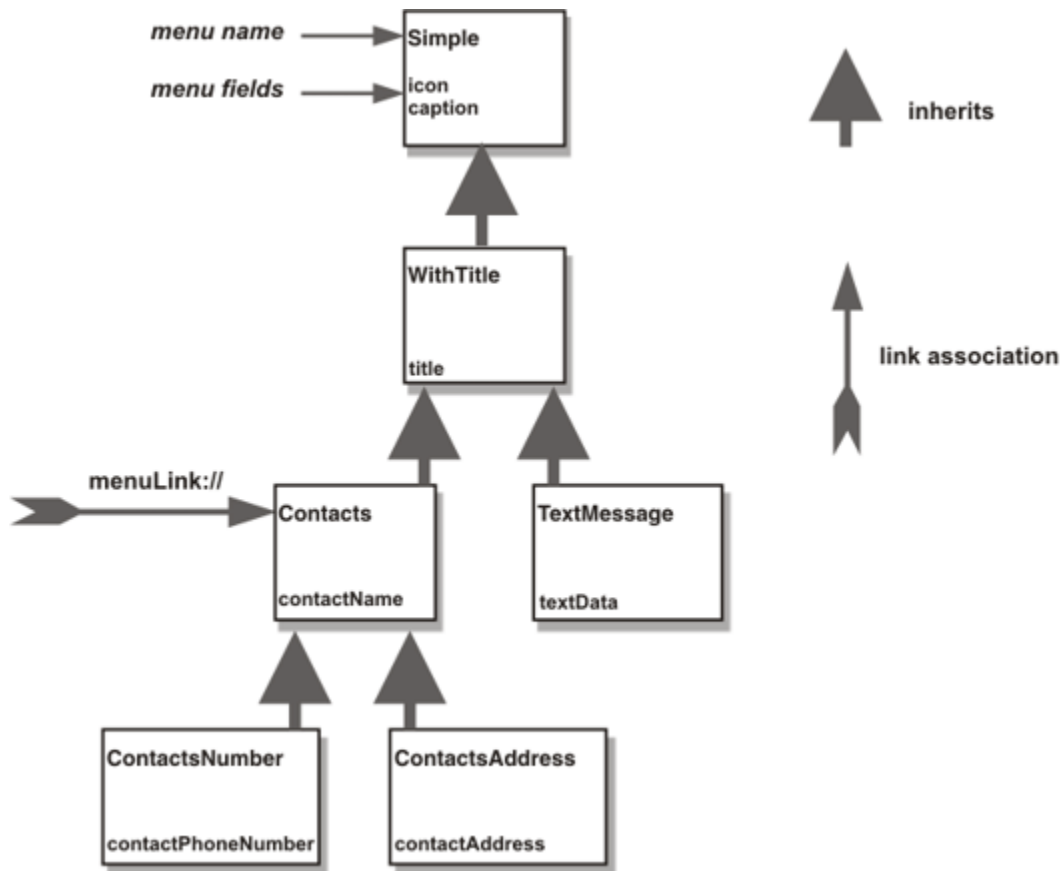


Note

A menu node cannot have more than one parent node; “multiple inheritance” is not allowed.

The following diagram shows an example menu hierarchy, with a menu link attached:

Figure 5-3. Example Menu Hierarchy



This would be represented by the following XML in the module definition file:

```
<module>
  <menus>
    <menu name="Simple">
      <fields>
        <field name="icon" dataType="string" mode="output"
previewValue="{icon1}" classType="graphic" context="item"/>
        <field name="caption" dataType="string" mode="output"
previewValue="caption" classType="text" context="item"/>
      </fields>
    </menu>
    <menu name="WithTitle" inherits="Simple">
      <fields>
        <field name="title" dataType="string" mode="output"
previewValue="Title" classType="text" context="menu"/>
      </fields>
    </menu>
    <menu name="Contacts" inherits="WithTitle">
      <fields>
        <field name="contactName" dataType="string" mode="output"
previewValue="Name" classType="text" context="item"/>
      </fields>
    </menu>
  </menus>
</module>
```

```
<menu name="TextMessage" inherits="WithTitle">
  <fields>
    <field name="textDate" dataType="time" mode="output"
previewValue="01/01/2000" classType="text" context="item"/>
  </fields>
</menu>
<menu name="ContactsNumber" inherits="Contacts">
  <fields>
    <field name="contactNumber" dataType="string" mode="output"
previewValue="+44 01 222 4444" classType="text" context="item"/>
  </fields>
</menu>
<menu name="ContactsAddress" inherits="Contacts">
  <fields>
    <field name="contactAddress" dataType="string" mode="output"
previewValue="22 Acacia Avenue" classType="text" context="item"/>
  </fields>
</menu>
</menus>
<links>
  <link linkPrefix="menuLink" linkType="menu" menu="Contacts"/>
</links>
</module>
```

The following functions would have to be implemented by the module writer:

```
/* Startup / shutdown */

IFX_RETURN_STATUS IFXM_ifxuidemo_Initialize(
    IFX_HMODULEID      hModuleId,
    IFX_HMODULE        *phModule);

IFX_RETURN_STATUS IFXM_ifxuidemo_ShutDown(
    IFX_HMODULE        hModule);

/* Menus */

IFX_RETURN_STATUS IFXM_ifxuidemo_OpenMenu_menuLink(
    IFX_HMODULE        hModule,
    IFX_HMENUID        hMenuId,
    IFX_HMENU          *phMenu);

IFX_RETURN_STATUS IFXM_ifxuidemo_CloseMenu_menuLink(
    IFX_HMODULE        hModule,
    IFX_HMENU          hMenu);

IFX_RETURN_STATUS IFXM_ifxuidemo_GetItemCount_menuLink(
    IFX_HMODULE        hModule,
    IFX_HMENU          hMenu,
    IFX_INT32          *pCount);

IFX_RETURN_STATUS IFXM_ifxuidemo_GetFirstActiveItem_menuLink(
    IFX_HMODULE        hModule,
    IFX_HMENU          hMenu,
    IFX_INT32          *pItem);

IFX_RETURN_STATUS IFXM_ifxuidemo_SetActiveItem_menuLink(
```

```

        IFX_HMODULE      hModule,
        IFX_HMENU        hMenu,
        IFX_INT32        itemIndex);

IFX_RETURN_STATUS IFXM_ifxuidemo_GetSortFieldSize_menuLink(
    IFX_HMODULE      hModule,
    IFX_HREQUEST     *phRequest,
    IFX_HMENU        hMenu,
    IFX_INT32        *pSize);

IFX_RETURN_STATUS IFXM_ifxuidemo_GetSortDirection_menuLink(
    IFX_HMODULE      hModule,
    IFX_HMENU        hMenu,
    IFX_SORT_DIRECTION *pDirection);

IFX_RETURN_STATUS IFXM_ifxuidemo_GetSortType_menuLink(
    IFX_HMODULE      hModule,
    IFX_HMENU        hMenu,
    IFX_SORT_TYPE     *pType);

IFX_RETURN_STATUS IFXM_ifxuidemo_GetLoadOnDemand_menuLink(
    IFX_HMODULE      hModule,
    IFX_HMENU        hMenu,
    IFX_INT32        *pLoadOnDemand);

IFX_RETURN_STATUS IFXM_ifxuidemo_GetPlaceholderSize_menuLink(
    IFX_HMODULE      hModule,
    IFX_HREQUEST     *phRequest,
    IFX_HMENU        hMenu,
    IFX_INT32        *pSize);

IFX_RETURN_STATUS IFXM_ifxuidemo_GetMenuType_menuLink(
    IFX_HMODULE      hModule,
    IFX_HMENU        hMenu,
    IFX_MENU_TYPE_IFXUIDEMO_MENULINK *pType);

IFX_RETURN_STATUS IFXM_ifxuidemo_GetSortFieldData(
    IFX_HMODULE      hModule,
    IFX_HREQUEST     hRequest,
    IFX_WCHAR        *szData);

IFX_RETURN_STATUS IFXM_ifxuidemo_GetPlaceholderData(
    IFX_HMODULE      hModule,
    IFX_HREQUEST     hRequest,
    IFX_WCHAR        *szData);

/* Field Data */

IFX_RETURN_STATUS IFXM_ifxuidemo_GetFieldStringSize_menuLink_icon(
    IFX_HMODULE      hModule,
    IFX_HREQUEST     *phRequest,
    IFX_HMENU        hMenu,
    IFX_INT32        item,
    IFX_INT32        *pSize);

IFX_RETURN_STATUS IFXM_ifxuidemo_GetFieldStringSize_menuLink_caption(
    IFX_HMODULE      hModule,
    IFX_HREQUEST     *phRequest,

```

```

        IFX_HMENU          hMenu,
        IFX_INT32          item,
        IFX_INT32          *pSize);

IFX_RETURN_STATUS IFXM_ifxuidemo_GetFieldStringSize_menuLink_title(
        IFX_HMODULE        hModule,
        IFX_HREQUEST       *phRequest,
        IFX_HMENU          hMenu,
        IFX_INT32          item,
        IFX_INT32          *pSize);

IFX_RETURN_STATUS IFXM_ifxuidemo_GetFieldStringSize_menuLink_contactName(
        IFX_HMODULE        hModule,
        IFX_HREQUEST       *phRequest,
        IFX_HMENU          hMenu,
        IFX_INT32          item,
        IFX_INT32          *pSize);

IFX_RETURN_STATUS
        IFXM_ifxuidemo_GetFieldStringSize_menuLink_contactAddress(
        IFX_HMODULE        hModule,
        IFX_HREQUEST       *phRequest,
        IFX_HMENU          hMenu,
        IFX_INT32          item,
        IFX_INT32          *pSize);

IFX_RETURN_STATUS
        IFXM_ifxuidemo_GetFieldStringSize_menuLink_contactNumber(
        IFX_HMODULE        hModule,
        IFX_HREQUEST       *phRequest,
        IFX_HMENU          hMenu,
        IFX_INT32          item,
        IFX_INT32          *pSize);

IFX_RETURN_STATUS IFXM_ifxuidemo_GetFieldStringData(
        IFX_HMODULE        hModule,
        IFX_HREQUEST       hRequest,
        IFX_WCHAR          *szData);

```

The following enum provides the options for the implementation of IFXM_ifxuidemo_GetMenuType_menuLink:

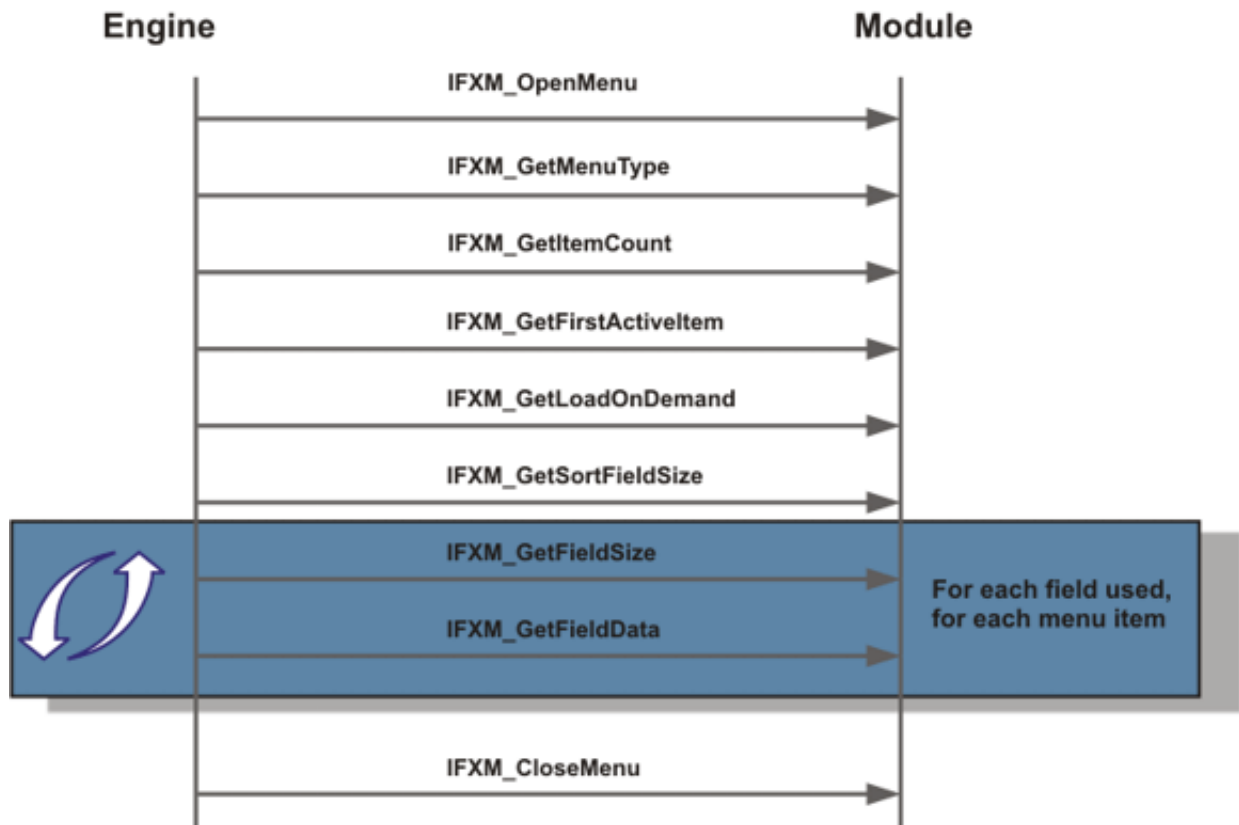
```

typedef enum
{
    IFX_MENU_TYPE_IFXUIDEMO_MENULINK_CONTACTS,
    IFX_MENU_TYPE_IFXUIDEMO_MENULINK_CONTACTSADDRESS,
    IFX_MENU_TYPE_IFXUIDEMO_MENULINK_CONTACTSNUMBER
} IFX_MENU_TYPE_IFXUIDEMO_MENULINK;

```

Figure 5-4 shows the sequence of calls used when opening a dynamic menu for a simple menu (in which “load on demand” is not used, and sorting is not applied to the menu items).

Figure 5-4. Dynamic Menu Call Sequence



Modules Supporting Integrated Applications

For each plugin element declared in the module definition file, a set of module functions are defined to create, activate, position, deactivate, interact with, and destroy an instance of the element. The precise set of functions varies with the element properties (for instance, is the element a graphic element or event handler? Does the element support stylus interaction?).

Elements are created via a standard Inflexion UI URI-style link. The link is defined in the “elements” section of the module definition file, rather than in the links section, to emphasize the uniqueness of this sort of link (they should only be used as “resource” links within the UI theme, and not as functional links).

A simple element that has a screen presence, and can handle stylus events, is defined in the module definition file as illustrated here:

```
<elements>
  <element type="graphic" linkPrefix="createSampleElement"
  supportStylus="true"/>
</elements>
```

The Integrator tool generates the following functions for this element:


```

/* Element functions */

IFX_RETURN_STATUS IFXM_ifxuisimpleedit_CreateElement_createSampleElement (
    IFX_HMODULE      hModule,
    IFX_HELEMENTID   hElementId,
    IFX_HELEMENT     *phElement,
    IFX_HMENUID      hMenuId,
    IFX_HMENU        hMenu,
    IFX_INT32        item,
    IFX_ELEMENT_PROPERTY *pProperty);

IFX_RETURN_STATUS
IFXM_ifxuisimpleedit_DestroyElement_createSampleElement (
    IFX_HMODULE      hModule,
    IFX_HELEMENT     hElement);

IFX_RETURN_STATUS
IFXM_ifxuisimpleedit_ActivateElement_createSampleElement (
    IFX_HMODULE      hModule,
    IFX_HELEMENT     hElement);

IFX_RETURN_STATUS
IFXM_ifxuisimpleedit_DeactivateElement_createSampleElement (
    IFX_HMODULE      hModule,
    IFX_HELEMENT     hElement);

IFX_RETURN_STATUS
IFXM_ifxuisimpleedit_ProcessElementKeyUpEvent_createSampleElement (
    IFX_HMODULE      hModule,
    IFX_HELEMENT     hElement,
    IFX_INT32        key);

IFX_RETURN_STATUS
IFXM_ifxuisimpleedit_ProcessElementKeyDownEvent_createSampleElement (
    IFX_HMODULE      hModule,
    IFX_HELEMENT     hElement,
    IFX_INT32        key,
    IFX_INT32        *pConsumed);

IFX_RETURN_STATUS
IFXM_ifxuisimpleedit_GetElementCaretPosition_createSampleElement (
    IFX_HMODULE      hModule,
    IFX_HELEMENT     hElement,
    IFX_INT32        *pIndex);

IFX_RETURN_STATUS
IFXM_ifxuisimpleedit_SetElementFocus_createSampleElement (
    IFX_HMODULE      hModule,
    IFX_HELEMENT     hElement);

IFX_RETURN_STATUS
IFXM_ifxuisimpleedit_UnsetElementFocus_createSampleElement (
    IFX_HMODULE      hModule,
    IFX_HELEMENT     hElement);

IFX_RETURN_STATUS
IFXM_ifxuisimpleedit_PositionElement_createSampleElement (
    IFX_HMODULE      hModule,

```

```

        IFX_HELEMENT          hElement,
        IFX_POSITION          *pPosition);

IFX_RETURN_STATUS IFXM_ifxuisimpleedit_PaintElement_createSampleElement (
        IFX_HMODULE          hModule,
        IFX_HELEMENT          hElement);

IFX_RETURN_STATUS
        IFXM_ifxuisimpleedit_ChangeElementMode_createSampleElement (
        IFX_HMODULE          hModule,
        IFX_HELEMENT          hElement,
        IFX_ELEMENT_MODE      newMode,
        void                  *pContext);

IFX_RETURN_STATUS
        IFXM_ifxuisimpleedit_ProcessElementStylusHitTest_createSampleElement (
        IFX_HMODULE          hModule,
        IFX_HELEMENT          hElement,
        IFX_INT32             x,
        IFX_INT32             y,
        IFX_INT32             *pHit);

IFX_RETURN_STATUS

IFXM_ifxuisimpleedit_ProcessElementStylusDragEvent_createSampleElement (
        IFX_HMODULE          hModule,
        IFX_HELEMENT          hElement,
        IFX_INT32             x,
        IFX_INT32             y);

IFX_RETURN_STATUS
        IFXM_ifxuisimpleedit_ProcessElementStylusUpEvent_createSampleElement (
        IFX_HMODULE          hModule,
        IFX_HELEMENT          hElement,
        IFX_INT32             x,
        IFX_INT32             y);

IFX_RETURN_STATUS

IFXM_ifxuisimpleedit_ProcessElementStylusDownEvent_createSampleElement (
        IFX_HMODULE          hModule,
        IFX_HELEMENT          hElement,
        IFX_INT32             x,
        IFX_INT32             y,
        IFX_INT32             *pConsumed);

IFX_RETURN_STATUS

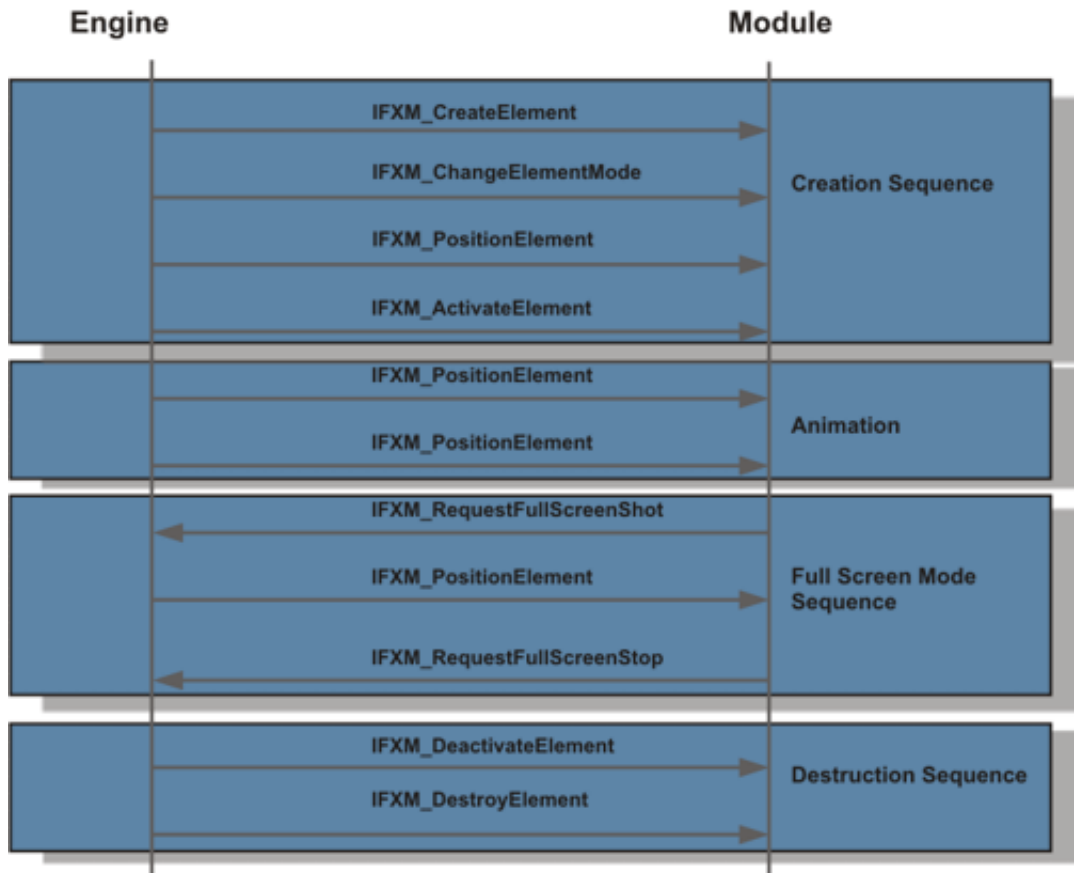
IFXM_ifxuisimpleedit_ProcessElementStylusCancelEvent_createSampleElement
(
        IFX_HMODULE          hModule,
        IFX_HELEMENT          hElement);

```

Figure 5-5 indicates the pattern of IFXM calls during the lifecycle of an element instance, when the element is used in direct rendering mode. Elements using buffered mode have a similar call sequence. The difference is that they do not get position calls (the animation is all done within the Inflexion UI engine), but they do receive a sequence of paint plugin calls so that they can

integrate with the Inflexion UI Runtime component paint cycle (it is the responsibility of the module to notify the Runtime component whenever the contents of the buffer have changed, via the IFXI_RequestRefreshBufferedElement method). Note also that buffered mode elements cannot go “full screen”.

Figure 5-5. Direct Mode Plug-in Elements



Module Definition File

The module definition file is an XML file that describes the functionality the module offers the Inflexion UI Runtime component. The UI Integrator tool uses it to generate the Integration Layer that is positioned between the engine and the set of modules to be used. When the UI Integrator tool parses a module definition file, another output is a header file detailing the API that the module must implement to support the Integration Layer. By convention, module definition files have a *.module* extension.

Format

- Module Name

The **module** element takes a **name** attribute. The value of this attribute scopes the supported features and should be unique within the system being assembled to avoid potential name clashes. The Integrator tool will terminate with an error in this situation.

- Exclusivity

The **module** element can contain one attribute in addition to **name**: **exclusivitySupport**. This should be set to “true” if the module requires exclusive access to the Inflexion UI Runtime component. This might be necessary to prevent other modules from displaying menus. For example, in the case of an incoming telephone call, the note informing you of the call and prompting you to accept or reject the call should not be overlaid by any other menu. This requires the module to implement a set of exclusivity-related methods (see the [Module Integration API Reference](#) chapter). If this attribute is set to “false” or is absent, the module does not require implementing any exclusivity methods.

Fields

The set of global fields (for example, fields that do not vary according to menu or item context) supported by the module are listed in the **fields** element within the **module** element. The **fields** element has one or more field elements. Each field element must have a name attribute that identifies it from within the UI theme when surrounded by the field token delimiters (that is, “[name]”). The name chosen must be unique to all modules used in the system.

Each **field** element can also have a **dataType** attribute specifying whether the raw field data is of type **string**, **int**, **float**, or **time**, or **boolean**. If **dataType** is missing, the field is assumed to be of type **string**.

An optional attribute of the field element, **defaultValue** can be used to define the string returned if the data mapping did not return a specific value. This default string can contain one format specifier (for example, %d,%s) to incorporate the raw value returned by the module API function. In the case of **time**-type field data, the **defaultValue** is always used and can contain any of the formatting strings associated with the `strftime` C run-time library function. If **defaultValue** is missing, the Inflexion UI Runtime component uses the following formats:

- "%s" - string type
- "%d" - int, boolean types
- "%f" - float type
- "%H:%M %d/%m/%y" - time type

In the Inflexion UI, module fields fall into one of two categories: either fields that are modified by the user via the UI (for example, if the field is modified by a drag region element), or fields that are modified by the device and reported to the user via the UI. The field element attribute **mode** states which category the field falls into - either **input** if the field is modified by the UI, or **output** if the field is modified by the module. If this attribute is missing, the field is assumed to be an output field. “string” type fields may not be “input” type fields.

If the field mode attribute is input, then the maximum and minimum allowable values for **int** and **float** type fields can be set using the `maxValue` and `minValue` attributes. These assist the theme writer to set up valid maximum and minimum limits when attaching these fields to displacements or touch region elements.

The attribute `editType` defines whether the field can be edited using a common control, such as the Simple Edit. If set, prototypes for the buffer sharing IFXM functions are generated for this field. Currently, only **string** is supported, and that the `dataType` must also be **string** for this to work.

Each field can also have `previewValue` and `classType` attributes, the former is a string value displayed in the Inflexion UI Express when it encounters use of field token in a theme (the Express cannot integrate directly with the module, obviously), and the latter is a hint to the Express when the field represents some resource to be loaded into a UI Theme element dynamically (for example, an icon resource to be used as the image for a graphic element). Express uses this information to help the theme designer select appropriate fields for the elements they are adding to the theme.

Links

If a module supports additional link types used in the Inflexion UI themes deployed on the target device, the module definition file must have a **links** element. This element contains one or more **link** elements. Each **link** element must have a **linkPrefix** attribute that indicates the link prefix used within the theme XML. This consists solely of ASCII alphanumeric characters and the “_” character. It should be as specific as possible to avoid name clashes with the link prefixes other modules provide. In addition to the **linkPrefix** attribute, the **link** element can have a **bodyType** attribute (indicating the type of the data in the link body - **int**, **float**, **time**, **boolean**, or **string**). Also, if **bodyType** is used, then the **defaultBody** attribute provides a default body value to be used when no value is provided in the activated link URI.

Each **link** element can have zero or more **param** elements representing the parameters that can be used following the link body. Each **param** element must have a **name** attribute, a **type** attribute (one of **int**, **float**, **time**, **boolean**, **string**). It can have a **defaultValue** attribute indicating the value to be used when the parameter is not present in the activated link URL. Links can do one of three things:

- Perform a synchronous action on the device
- Perform an asynchronous action on the device
- Open a dynamic menu supported by the module

The type of the link is specified using the `linkType` attribute, which should be set to `functionSync`, `functionAsync` or `menu` respectively for these three cases.

If **linkType** is set to **menu**, then you must also specify what node on the menu tree this link is attached to. This is done by setting the attribute `menu` to be one of the menu name attributes

from the menus section. Note that this association indicates to the theme designer that this link can open a menu with name equal to the references node, or any node that inherits from the node, and that the set of fields available for use on an attached menu component template is the union of all the fields for the attached menu node, all the nodes that inherit from that node and for the ancestor nodes up to the root node.

Menus

If the module supports one or more dynamic menus (that is, menu components whose items are populated by the module to reflect the situation at the current moment in time), then the module definition file must contain a **menus** element, containing one or more **menu** elements. Each **menu** element represents one menu type supported by the module and must have a **name** attribute. Additionally, each menu can specify one other menu entry from which it “inherits” all the fields defined by the parent menu. This is done by setting the **inherits** attribute to the **name** of the menu to use as the parent. This is how the menu structure discussed in the [Modules Supporting Dynamic Menus](#) topic is created. Each menu item can have one **fields** element.

The fields that populate the dynamic menu are defined by the **fields** element, which must have one or more **field** elements. Each **field** element follows the same rules as for the global fields laid out in the [Fields](#) section, above. The only difference between fields at the global level and the fields at menu level is that menu fields can have menu or item scope (for example, the value of the field can change for each menu instance, or for each item instance in the menu). There is an additional attribute over the ones for global fields, **context**, that should be set to **menu** for fields that have menu scope and **item** for fields with item scope.

Elements

If a module supports one or more plugin elements, then the module definition file must contain an **elements** element, containing one or more **element** elements. Each **element** element must have a **type** attribute, indicating whether the element is a “graphic” or an “eventHandler” element and a **linkPrefix** attribute indicating the element creation link prefix. As with other URI link descriptions in the module definition file, each **linkPrefix** attribute can be augmented by **bodyType** and **defaultBody** attributes (see the [Links](#) section for further details). The **linkType** attribute is ignored in this case (all element creation links are considered synchronous function links). The **element** element can augment the creation link definition with zero or more **param** elements, each having **name**, **type**, and **defaultValue** attributes (see the [Links](#) section for more information). Two other optional attributes exist for the **element** element: **supportFields**, and **supportStylus**. **supportFields** must be set to **true** if the element can expand field data that is not owned by the module (the example here is the SimpleEdit module, which must be able to edit any field). If **supportFields** is set to **true**, the module must implement the version of GetFieldStringSize/GetFieldData that accepts an element handle. This function is called any time a token is expanded for an engine element with a plugin handle of this element type attached. The function is called before the global GetFieldStringSize/GetFieldData methods. If **supportStylus** is set to **true**, the element is allowed to receive and interpret finger/stylus events. This necessitates the implementation of a

set of pointer-related methods for this module element. If **supportStylus** is missing, the element does not receive pointer events.

Triggers

The module can attempt to have the Inflexion UI Runtime component perform an action through any number of module-specific “triggers”. These only have an effect if the loaded the Inflexion UI responds to the triggers. If the module supports this feature, then a **triggers** element must be added to the module definition file containing one or more **trigger** elements. Each **trigger** element must have a **name** attribute, which should be a “friendly” name for the trigger used within the Inflexion UI Express. Each trigger must also have a **key** attribute indicating the trigger to use. Any alphanumeric string can be used with the “special” key strings “select”, “back”, “up”, “down”, “left”, “right”, “#”, and “*”. Platform-specific key scan codes can also be used by a string in the form of “scan_N” where “N” is the decimal numeric scan code (for example, “scan_962” is equivalent to the button with scan code 0x03C2 being pressed on the device).

Each **trigger** element can also have a **description** attribute that contains information for the theme designer to indicate the action module expects the trigger to perform within the UI. It is recommended that this optional attribute be used in most trigger examples.

SimpleEdit Module Example

The following example is taken from the module definition file for the SimpleEdit module included with the Inflexion UI Runtime component. It can be found in the following location: *nucleus/os/ui/inflexionui/modules/simpleedit/ifxui_simpleedit.module*.

```
<module name="ifxui_simpleedit">
<fields>
  <field name="simpleEditInputMode" previewValue="{autoCapWord}"
    dataType="int" defaultValue="{nonInputMode}" >
</field>
</fields>
```

This module contains a single global field token, **simpleEditInputMode**, that can be used within a theme to indicate the current edit mode. The theme designer can replace the tokenized returned strings.

The demo theme uses the [simpleEditInputMode] field token as the resource string for a graphical element in the module menu’s template (not available with the Inflexion UI Runtime component application demonstration; the original theme is shipped with the UI Express). Load the demonstration application on the display and navigate to the “Integration Menu” item. The icon in the top-right corner is selected, using the expansion of [simpleEditInputMode]. Tapping this icon with the stylus (or pressing a key mapped to the # key) changes the input mode, causes the SimpleEdit module to notify the engine that the value of [simpleEditInputMode] has changed, and regenerates the icon based on the new value of [simpleEditInputMode].

```
<elements>
  <element type="eventHandler" linkPrefix="simpleEdit"
bodyType="string" supportFields="true">
    <param name="style" type="string" />
    <param name="maxLength" type="int" defaultValue="10" />
    <param name="inputMode" type="string" />
    <param name="caretPosition" type="int" defaultValue="-1" />
  </element>
</elements>
```

The SimpleEdit module supports a single element class of type **eventHandler** (that is, it has no direct on screen presence, but handles events on behalf of the engine entity to which it is attached). Instances of this element are created using the **eventHandler** attribute for the **resource** element in the theme for a particular text element, using the “simpleEdit” link prefix in the **eventHandler** link URI. The link body is the field being edited. There are four possible parameters indicating the style, max length, initial edit mode, and initial carat position for this editable field.

The module must implement a set of functions that allow the element instance to be created and destroyed and to handle events passed on to the element by the Inflexion UI Runtime component. Each event that alters the string buffer that SimpleEdit associates with the engine entity for which it is handling events, causes SimpleEdit to notify the Runtime component of the change via a call to `IFXI_RequestFieldRefresh`.

```
<links>
  <link linkPrefix="cycleInputMode" />
</links>
```

The module supports a single link type, **cycleInputMode**, which advances the current input mode to the next mode in the sequence (that is, from all lowercase to all uppercase to capitalized words, and so on). While in the demo theme, tapping the stylus on the input mode icon in the Integration Menu triggers the link URI `cycleInputMode://`. This is handled by the SimpleEdit module in `IFXM_ifxuisimpleedit_ExecuteLink_cycleInputMode`.

Device Definition File

The device definition file (*.device) is a manufacturer-specific repository of device capabilities used to generate the integration layer for an application and package UI themes ready for deployment to their devices. Each manufacturer’s device that includes the Inflexion UI Runtime component must have a separate device definition file.

Portable Key Codes

Portable Key Codes allow a theme to be used across a variety of handsets, irrespective of the actual scan code associated with the hardware on a particular handset. Portable Key Codes allow you to give a common name to each key and define the scan code separately for a device in the device definition file. This allows the theme designer to abstract a template away from any particular device, since the template references the portable key code, rather than a

particular scan code, when setting up triggers, and the device definition file resolves those portable key codes to particular scan codes. It is still possible, of course, for theme designers to use hardware-specific key codes in the template files, but this makes porting themes using the template to other devices a non-trivial task.

In the Inflexion UI Express, the Portable Key Codes defined in the device definition file appear in the list of available triggers and also appear as additional buttons in the previewer controls.

**Note**

The portable key code names are only used within the template - when calling the IFXE_Key_Up/IFXE_Key_Down functions, or when a module asks for a trigger to be executed via the IFXI_RequestTriggerKey function the device-specific hardware scan code must be used.

The Portable Key Codes feature is backward compatible with the existing pre-defined keys and the old scan_xxxx format of defining triggers.

Format

The device definition file is an XML file containing one *deviceDefinition* element. The *deviceDefinitions* element contains the *graphicalCapabilities*, *displays*, *keys*, *languages*, *displayModes* and *designSizes* elements that define the set capabilities for this device (see the following sections for more details on these elements).

The *deviceDefinition* element must have the *name* attribute set to a string that indicates the device and an optional *description* attribute containing extended information about the platform. This aids the theme designer when choosing a target device.

Graphical Capabilities

The graphicalCapabilites element has a single attribute:

- **lightCount** -

This indicates the number of OpenGL ES lights available on the OpenGL ES implementation on the device.

There is one child element, **maximumTextureDimensions**, with width and height attributes giving the largest valid texture size for the OpenGL ES implementation on the device.

Displays

The **displays** element must contain one or more **display** elements, each having a **name** attribute and optional **description** and **defaultMode** attributes. The value of the **name** attribute should be informative, for example, a clamshell form factor device might have a “Primary” screen and

a “Cover” screen. Each **display** element must contain one or more **support** elements, each with a **mode** attribute. The value of the **mode** attribute should match one of the display mode names from the **displayModes** element. The mode specified in the **defaultMode** attribute is used if the application pointing to this file does not specify a preferred mode. If the **defaultMode** is missing, the first mode listed in the support element is used.

Keys

The **keys** element allows any special hardware keys available on the device to be declared and used by a theme designer. “Special” refers to keys other than the pre-defined set of the Inflexion UI keys: “select”, “back”, “up”, “down”, “left”, “right”, “#”, “*”, and “0-9”. The **keys** element must contain one or more **key** elements each with a **name** and optional **description** attribute. The **key** element should contain a **scanCode** attribute, indicating the (decimal) scan code of that key and an optional **simtestCode** for use with a *SimTest.hmi* file. If neither code is provided, the Inflexion UI Express interprets this key as being unavailable on the device so that a pre-defined set of keys can be selectively disabled for this device. If a key name appears in more than one device element, even when mapped to different scan codes, it is considered a Portable Key Code.

Languages

The set of languages supported by a particular platform are enumerated in the **languages** element each represented by a **language** element. Each **language** element must have a **name** attribute, which should be set to the appropriate two-character language code from the ISO 639-1 list. An optional **description** attribute can be used to give a more helpful name for the language to aid the theme designer.

Display Mode

The list of display modes available to the device is contained in the **displayModes** element, which must contain one or more **displayMode** elements. Each **displayMode** element must have a **name** and optional **description** attribute indicating what the mode is (for example, “QVGA_P”, “QVGA resolution, portrait mode”), a **width** and **height** attribute giving the display dimensions in pixels, and a **designSize** attribute. The value of the **designSize** attribute must match one of the **designSize** names (see [Design Sizes](#), below).



Caution

When a device is in a particular mode, all theme templates must match this design size, or the menu using the template will not open.

Design Sizes

The **designSizes** element defines a pool of design sizes used by UI theme templates, each contained in a **designSize** element. Each **designSize** element must have a **name** attribute that is

unique in the set of design sizes. This is used in the **displayMode** element to associate a design size with a particular display mode and in UI theme template files to identify the design size in which the template was constructed to be viewed. Additionally, each **designSize** element must have **width** and **height** attributes giving the design size dimensions in pixels and an **origin** attribute giving the vertical position of the origin of the template coordinate space from the top of the device screen. The **designSize** element can also have the optional **description** attribute to provide the theme designer with additional information about the design size.

Target-Specific Tool Definitions

As part of the packaging step—when user interface themes designed in the Inflexion UI Express tool are being prepared for use by the UI Runtime component—certain resources may be processed by custom command-line-based tools. For example, you might want to pre-compile Open GL ES 2.0 shaders into binary objects to improve system start up time in production models, or you might want to process image data into compressed texture format, pre-mipmap image data, or use a non-lossy image packing format for particularly complicated images where the quantization used in the internal Inflexion UI image packaging format results in noticeable image artifacts.

The tools available for a particular target must be specified in the device definition file, which will result in themes that reference the device definition file being able to select those tools to be used when packaging images or shaders either on a per-file basis or across the whole set of relevant files.

The first step is to ensure that there is a **targetConfigurations** element as a child of the **deviceDefinitions** element. The **targetConfigurations** element can have one or more **targetConfiguration** elements, each one having a **name** attribute that should be something suitably meaningful that a theme designer can understand what the target is and when it should be selected. The **targetConfiguration** element then has a child element called **translationTools**, with one or more child elements called **translationTool**. Each translation tool describes either an image translation tool or a shader compilation tool, and there is no rule against having the same executable represented multiple times, each time with a slightly different command line provided that each **translationTool** has a unique **toolName** attribute that gives the tool configuration a meaningful title.

Each **translationTool** element must have a **toolType** attribute in addition to the **toolName**. Valid values are **shaderCompiler** for tools that compile Open GL ES shader files into binary objects, and **imageFileConverter** for tools that convert *.png* files into some other format. The tool can be given a description, useful for indicating the purpose of the tool entry, and the name of an executable must be given. Note that because the packager process will call out to these executables during packaging from various working directories, this executable must either be the full path to the executable, OR the executable location must be in the relevant PATH environment variable. To verify, open a command prompt and paste in the exact string used in the **executable** attribute and press **Enter**. the executable must be found, or the packaging step will fail if this tool is used.

Together with the executable attribute, a **body** attribute must be specified. This contains all the command line options along with placeholders for the input file (or files in the case of a vertex and fragment shader pair to be turned into a unified binary program) and output file name (for shader compiler tools). These placeholders are:

- \$1 - first input file
- \$2 - second input file (unified shader compilation case only, in which case \$1 is the vertex shader, and \$2 is the fragment shader)
- \$0 - output file (typically for shader compilation only)

If the tool is for shader compilation and it produces a unified binary program from the vertex and fragment shader sources, the attribute **unifiedShaderOutput** should be set to **true**.

To allow for the possibility that separate shader compilation tools are needed to compile the vertex and fragment shaders for a particular target, an optional **executable2** attribute can be used. If this attribute is defined, this tool will be used to compile the fragment shader, rather than that in the **executable** attribute. If this is missing or empty, the executable named in the **executable** attribute will be used. In the separate binary object shader compilation case it's normal that the command line is different between the vertex and fragment shader compilation, whether there are one or two tools used. For this reason, the command line to be used for compiling the fragment shader should be placed in the **body2** attribute, with the placeholder **\$2** indicating where the name of the fragment shader source file should go.

Finally, for compiled shader objects, the value of the Open GL enumerated entry indicating the format of the compiled shader object must be set in the **compiledShaderFormat**. This will be sent to the UI Runtime component and used in the call to **glShaderFormat** when the shader is loaded into the OGL pipeline.

Device Definition File Example

The following example is the sample device definition file located in the demo source code in the */demo/inflexionui_default* folder.

```
<?xml version="1.0" encoding="UTF-8"?>
<deviceDefinition name="Sample device" description="Sample device
definition." xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="inflexion_2.0/deviceDefinition.xsd">
  <supportedLanguages>
    <support language="en"/>
    <support language="cn"/>
  </supportedLanguages>
  <displays>
<display name="Primary" defaultMode="QVGA_P">
  <support mode="QVGA_P"/>
  <support mode="QVGA_L"/>
  <support mode="VGA_P"/>
  <support mode="VGA_L"/>
  <support mode="WVGA_P"/>
```

```

    <support mode="WVGA_L" />
    <support mode="QCIFP_P" />
    <support mode="QCIFP_L" />
    <support mode="QCIFS_P" />
    <support mode="QVGAS_P" />
  </display>
</displays>
<languages>
  <language name="en" description="English" />
  <language name="cn" description="Chinese" />
</languages>

<displayModes>
  <displayMode name="QVGA_P" width="240" height="320" designSize="QVGA_P"
description="QVGA Portrait" />
  <displayMode name="QVGA_L" width="320" height="240" designSize="QVGA_L"
description="QVGA Landscape" />
  <displayMode name="VGA_P" width="480" height="640" designSize="QVGA_P"
description="VGA Portrait" />
  <displayMode name="VGA_L" width="640" height="480" designSize="QVGA_L"
description="VGA Landscape" />
  <displayMode name="WVGA_P" width="480" height="800" designSize="WVGA_P"
description="WVGA Portrait" />
  <displayMode name="WVGA_L" width="800" height="480"
designSize="WVGA_L" description="WVGA Landscape" />
  <displayMode name="QCIFP_P" width="176" height="220"
designSize="QVGA_P" description="QCIF+ Portrait" />
  <displayMode name="QCIFP_L" width="220" height="176"
designSize="QVGA_L" description="QCIF+ Landscape" />
  <displayMode name="QCIFS_P" width="176" height="194"
designSize="QCIFS_P" description="QCIF+ Short Portrait" />
  <displayMode name="QVGAS_P" width="240" height="260"
designSize="QVGAS_P" description="QVGA Short Portrait" />
</displayModes>
<designSizes>
  <designSize name="QVGA_P" width="240" height="320" origin="160"
description="QVGA Portrait" />
  <designSize name="QVGA_L" width="320" height="240" origin="120"
description="QVGA Landscape" />
  <designSize name="WVGA_P" width="480" height="800" origin="400"
description="WVGA Portrait" />
  <designSize name="WVGA_L" width="800" height="480" origin="240"
description="WVGA Landscape" />
  <designSize name="QCIFS_P" width="176" height="194" origin="97"
description="QCIF+ Short Portrait" />
  <designSize name="QVGAS_P" width="240" height="260" origin="130"
description="QVGA Short Portrait" />
</designSizes>
<targetConfigurations>
  <targetConfiguration name="Hardware 1">
    <translationTools>
      <translationTool toolName="texture_compress low"
toolType="imageFileConverter" description="This method compresses
textures with low compression rate" executable="texcomp" body="-i $1 -rate
1" />
    </translationTools>
  </targetConfiguration>
</targetConfigurations>

```

```

    <translationTool toolName="texture_compress high"
    toolType="imageFileConverter" description="This method compresses
    textures with high compression rate" executable="texcomp" body="-i $1 -
    rate 2"/>
    <translationTool toolName="shader_compile unified"
    toolType="shaderCompiler" description="Open GL shader compilation tool
    producing a single, unified binary program" executable="shadercomp"
    body="-v $1 -f $2 -o $0" unifiedShaderOutput="true"
    compiledShaderFormat="32"/>
    <translationTool toolName="shader_compile separate"
    toolType="shaderCompiler" description="Open GL shader compilation tool
    producing two binary shader objects" executable="vertcomp" body="-i $1 -o
    $0" unifiedShaderOutput="false" executable2="fragcomp" body="-i $2 -o $0"
    compiledShaderFormat="16"/>
  </translationTools>
</targetConfiguration>
</targetConfigurations>
</deviceDefinition>

```

The languages supported by this device are English and Chinese, and the device supports all displayModes declared in this file.

The display modes offered in this platform definition file are the common phone handset screen resolutions QVGA and QCIF+, in both portrait and landscape orientations, along with the larger VGA resolutions and a short version of QCIF+ in portrait orientation. Each is tied to a design size listed in the next section. Note that QVGA design sizes are used for QCIF+ display modes. This causes all template coordinates to be scaled according to the ratio of the display mode screen width and the design size width. This is not guaranteed to enable QVGA themes to be viewed perfectly on a QCIF+ screen, but it avoids the need to rewrite templates.

Typically, the coordinate origin of a design size is half the height of the design size.

Application Definition File

Each application that uses Inflexion UI must have a corresponding application definition file (**.application*). The application definition file declares what modules the application supports (and thus what additional device-specific functionality is available to the theme writer) along with what entry points a theme for use with the application should provide (essentially landing pages for the application). Each application definition references a single device definition file, and is an input for both the theme packager utility and the UI Integrator utility.

Format

The application definition file contains:

- A single **applicationDefinition** element, with a compulsory **name** attribute (for the name of the application)
- A GUID attribute

- A **deviceDefinition** attribute containing the path to the device definition file for the device that will use this application
- An optional **packageName** for use with Android applications
- An optional **description** attribute containing more information about the application to aid the reader

The GUID attribute ties a packaged theme to the application with which it is designed to work. This ensures that a theme never uses extended functionality (from modules) that is unavailable on the deployed device. A theme cannot be loaded on a device whose GUID does not match the GUID embedded into the theme.

The application definition file also contains two child elements, one **supportedModules** element and an optional **entryPoints** element.

supportedModules

The **supportedModules** element lists all modules used by this application (the modules are therefore usable by the theme designer), in the form of one or more **support** elements, each with a **module** attribute. The value of this **module** attribute should be the path to a module definition file. The path can be relative to the application definition file's location. It can also be an absolute path, including networked paths. The path's maximum length is 255 characters.

Each support element can also have a **programmingLanguage** attribute, set to either **c** or **java**, indicating whether the module implementation is in C or Java (this affects what is output by the UI Integrator utility).

The **supportedModules** element can have an **enableCatchAll** attribute, set to **true** or **false**. If set to true, then the application has a module that will be queried if no other module can supply the requested data.

Note that the sample application supports four modules. One is the **SimpleEdit** module discussed in the [Module Definition File](#) topic, and the others are simple modules bundled with the default inflexionui demo source that support the population of the Integration Menu menu in the demo, drawing of a simple sine/cosine wave and one that supports the dynamic battery indicator in the sample theme.

entryPoints

The **entryPoints** element contains one or more **entryPoint** elements, each with a single **id** attribute. This must be a unique string identifier for a single entry point for the application. Applications can have multiple entry points (that are mapped to particular UI theme pages by the theme designer), or they can have none (in which case the theme determines the initial page to display).

Application Definition File Example

```
<applicationDefinition
name="Sample application"
GUID="ifxui_demo"
deviceDefinition="sample.device"
packageName="com.mentorgraphics.inflexionui.module"
description="Sample application definition."
>
  <supportedModules>
    <support module="Modules\demo\ifxui_demo.module"/>
    <support module="Modules\battery\ifxui_battery.module"/>
    <support module="Modules\plugin\ifxui_plugin.module"/>
    <support module="Modules\simpleedit\ifxui_simpleedit.module"/>
  </supportedModules>
  <entryPoints>
    <entryPoint id="rootPage"/>
  </entryPoints>
</applicationDefinition>
```

Mentor Embedded Inflexion UI Integrator

This command line tool generates an integration layer that sits between the Inflexion UI Runtime component and the set of modules that comprise the device. The modules are each described by module definition XML files; an application, as described in an application definition XML file, contains the set of modules to be used.

The Integrator tool generates the Integration Layer (in the form of a set of C or Java language files. Depending on whether the modules are written in C, Java, or both), that must be included in the application source.

Usage

```
UIIntegrator [/h] [/f] [/r] [/d defaultDir] [/x | /i incDir /j javaDir]
applicationFile
```

The required input for this command:

- applicationFile

The application definition XML file.

The command generates the following source code files:

- *ifxui_integration.c*

The main integration file. This routes messages and requests between the Inflexion UI engine and the modules.

- *[module-filename].h*

One per module listed. This header file describes the interface between the module and *ifxui_integration.c*. The module filename is based on the module definition XML filename, therefore *ifxui_simpleedit.module* produces *ifxui_simpleedit.h*, and so on.

The output files are placed in the current directory by default. This can be overridden using the following switches.

Table 5-2. Command Line Tool Switches

Switch	Description
/h	Shows the Help information.
/d	defaultDir sets a new default directory destination for the output files. This new default can be further overridden by the switches /x.
/f	Forces target files to be overwritten if they already exist. If this switch is not set, you are asked to confirm before any file is overwritten.
/i	incDir specifies the output directory for all output *.h files.
/j	javaDir specifies the output directory for all *.java files. This command overrides any /d setting for these files.
/r	Generate module files compatible with the Nucleus OS ReadyStart build system.
/x	This places the module header files in the same directories as their module definition files.

Integrator Example

The following regenerates the Integration Layer:

(*demo/inflexionui_default/src/ifxui_integration.c*) and the module headers (*demo/inflexionui_default/modules/demo/ifxui_demo.h*, *demo/inflexionui_default/modules/battery/ifxui_battery.h*, *demo/inflexionui_default/modules/plugin/ifxui_plugin.h* and *inflexionui/modules/commons/simpleedit/ifxui_simpleedit.h*), when run from the *inflexionui/tools* folder, overwriting all five files without user confirmation.

```
UI Integrator.exe /f /x /r /d ../../demo/inflexionui_default/src
../../demo/inflexionui_default/sample.application
```


Chapter 6

Porting Overview

This chapter details the Porting Layer—the graphics subsystem Inflexion UI Runtime uses to handle displaying frames on the screen, manipulation of screen data, and native font support. The Porting Layer and render mode configuration settings provide a means to port the Inflexion UI Runtime component to different graphics subsystems and fine tune the implementation so the Runtime component performs to the best of its ability on a chosen platform.

In this chapter:

Render Modes	67
Buffered Internal	67
Buffered Internal Compressed	68
Porting Layer	68
Memory	69
Canvas Management and Rendering	69
Native Fonts	73
Mutex Support	73
File System	73
Benchmarking	74
Custom Image Format Support	74

Render Modes

You can enable this mode by enabling the *render_direct_opengl* option in a ReadyStart configuration file.

You can enable Direct OpenGL 2.0 mode by enabling the *render_direct_opengl_20* option in a ReadyStart configuration file.

Buffered Internal

You can enable this mode by enabling the *render_internal* option in a ReadyStart configuration file.

The Inflexion UI engine uses a built-in 3D drawing mode.

Buffered Internal Compressed

You can enable this mode by enabling the *render_internal_compressed* option in a ReadyStart configuration file.

This mode uses the same internal rendering engine, but it keeps the graphics data in compressed form in memory. Compressed form means the graphics can only be drawn in 2D. 3D features are not supported in this mode. The compressed format improves the RAM footprint by a value of 20-30%. Image data is kept in compressed format, reducing animation frame rates by 20-30%.

Porting Layer

The Inflexion UI engine calls the porting API, which must be implemented in the Porting Layer you use. The Porting Layer is used in different ways depending on the render mode the engine uses. It is divided into the following seven sections:

- [General](#)
- [Memory](#)
- [Canvas Management and Rendering](#)
- [Native Fonts](#)
- [Mutex Support](#)
- [File System](#)
- [Benchmarking](#)
- [Custom Image Format Support](#)

General

The Porting Layer's General section pertains to initialization, shutdown, display configuration, and EGL utility functions.

Table 6-1. General Porting Layer API Functions

Function	Description
IFXP_Initialize	Allocates any resources required by the Porting Layer and prepares the Porting Layer for use.
IFXP_Shutdown	Frees any resources allocated by the Porting Layer and closes it down.
IFXP_Timer_Get_Current_Time	Retrieves the system time in millisecond station required based on the display mode string.

Table 6-1. General Porting Layer API Functions

Function	Description
IFXP_Timer_Schedule	Schedules a timer event to expire after the requested time (in milliseconds). When the timer expires, IFXE_Process_Timers should be called.
IFXP_Check_GL_Extension	Called in the Direct OpenGL 1.1 or Direct OpenGL 2.0 case to determine which extensions to the OpenGL specification are supported by the hardware driver.

Memory

Heap memory allocation by the Inflexion UI Runtime component is all handled by the porting layer. This helps porting the Runtime component to platforms with different memory model requirements. The size of the memory pool that the Inflexion UI Runtime will reserve for its own use is set in *nucleus/os/ui/inflexionui/porting/inc/ifxui_porting_nucleus.h*, by the value of the define `IFXP_ENGINE_MEM_POOL_SIZE`.

Inflexion for ReadyStart also makes use of the Nucleus “partition pool” feature, reserving a subsection of the Inflexion memory pool for small, fixed-size allocations. This helps speed up page opening and reduces fragmentation, at the expense of a slightly higher “minimum” memory usage figure. To disable this feature, uncomment the define `IFXP_MEM_USE_PARTITION_POOLS` in *nucleus/os/ui/inflexionui/porting/inc/ifxui_porting_cfg.h*. To modify the fraction of Inflexion pool used for this purpose, see the `PARTITION_FRACTION` define in *nucleus/os/ui/inflexionui/porting/src/ifxui_porting_memory.c*.

Table 6-2. Memory Allocation Functions

Function	Description
IFXP_Mem_Allocate	Allocate a block of memory of the requested size in bytes.
IFXP_Mem_Deallocate	Release a block of memory previously allocated with <code>IFXP_Mem_Allocate</code> .

Canvas Management and Rendering

Canvas management is only used by the buffered render modes.

Canvas management creates a foreground buffer into which the Inflexion UI Runtime component renders a frame and a buffer to cache the background image, if required.

The canvas’s color format is determined by ReadyStart configuration options (for example, `canvas_mode_8888`). In general, the canvas format should match the display format. However,

this is platform-specific. For example, if a platform has an RGB565 display but has hardware acceleration to convert from RGB888 to RGB565, it is more efficient for the Inflexion UI Runtime engine to operate in RGB888 mode and have the hardware perform conversion to RGB565 when it copies content from the canvas to the display. In this case, the `IFXP_Render_Display` function is implemented to perform the conversion.

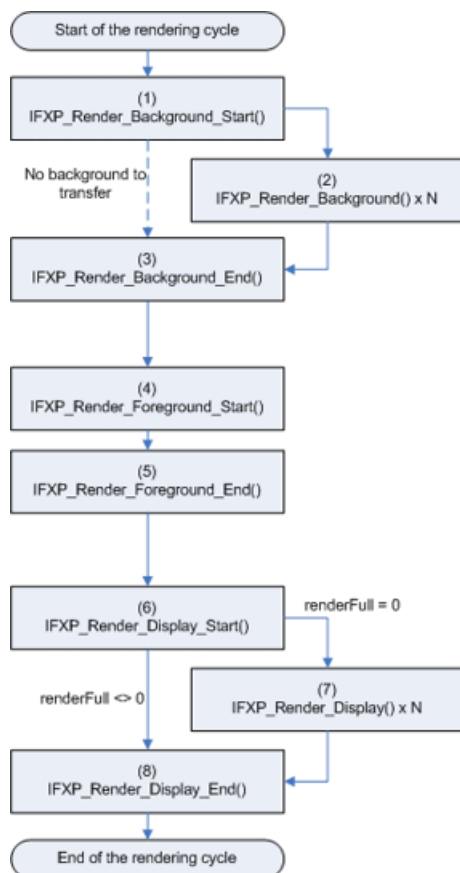
Another option for canvas management is to use a different memory for canvasses. For example, placing the main canvas in tightly coupled memory. In this case, the `IFXP_Canvas_Create` function is implemented to allocate memory from the TCM.

There are some methods to support the unpacking of image data that is packed in some other format to the proprietary Inflexion UI image file format. These are only necessary if the proprietary Inflexion UI image format is causing a problem for certain specific images, or the render mode is one of the Open GL ES render modes and compressed textures are to be used. See the [Target-Specific Tool Definitions](#) topic for more details.

Rendering API calls are only issued when the Inflexion UI engine operates in one of the buffered render modes. The rendering pipeline manipulates a background and foreground canvas to produce content that is displayed on the screen.

The rendering functions are called at various stages of the rendering pipeline. These should be implemented to perform the required operations using the graphics subsystem available on the platform. They also enable a hardware accelerated platform to start asynchronous operations and block waiting for an asynchronous operation to complete, where appropriate. The following diagram shows the flow of the rendering calls into the Porting Layer.

Figure 6-1. The Flow of Rendering Calls into the Porting Layer



1. This step must wait for any operations to complete. For example, if an asynchronous operation starts in step (8) of the previous frame, then you must block waiting for completion before continuing. If batch operations are supported, this indicates the start of a background batch.
2. This step copies sections of the background canvas to the foreground canvas. This is done to erase the region that is about to be rendered to. If layer blending is supported, the foreground canvas should be erased to transparency in this step, instead of performing a copy operation. This can be executed multiple times.
3. If batch operations are supported, this step sets the background batch running.
4. This step must block until any asynchronous operations on the foreground canvas have completed. If required, the canvas must be locked so that the Inflexion UI engine can write directly to the bitmap buffer.
5. This step unlocks the canvas if required.
6. The implementation of this step depends on the value of `renderFull` in the display structure.

If it is zero, this function starts the foreground batch. If it is non-zero, this function copies the whole canvas to the display and returns without executing step (7).

7. This step copies sections of the canvas to the display. This can be executed multiple times.
8. If a batch of operations are queued in step (7), this function sets the batch running. This step will NOT block waiting for completion - that is done in step (1).

Error Notes

The Porting Layer displays an error note under error conditions, for example, when the Inflexion UI engine's memory is full. In this case, the Porting Layer must have enough resources reserved to display the message without requiring additional memory.

The Porting Layer implementation is responsible for displays and allows the dismissal of the note if appropriate.

Native Fonts

Native fonts are used when `use_native_fonts` is enabled. The Porting Layer allows the Inflexion UI engine to use an externally implemented font library.

Table 6-3. Native Fonts

Function	Description
IFXP_Text_Open_Font IFXP_Text_Close_Font	Opens and closes a specific font by name. The remaining functions use the structure created by open to render a string using the opened font.
IFXP_Text_Get_Width IFXP_Text_Get_Char_Height	Calculates the size of a text canvas in pixels. This determines what is rendered by the Inflexion UI engine.
IFXP_Text_Canvas_Create IFXP_Text_Canvas_Destroy	Creates and destroys a single channel (alpha) bitmap buffer containing the text string supplied.

If the font engine does not support single channel rendering, the Porting Layer must manage the creation and conversion to a single channel bitmap. If `IFXP_FALLBACK_FONT_SUPPORT` is defined in the *ifxui_porting_cfg.h* file, then the font porting layer may use a set of system fonts to find any glyphs not supported by the font in use by the Inflexion UI theme when rendering text. The array `gFallBackFonts` in the file *ifxui_porting_font.c* should be edited to add the list of system fonts to be used, in the order that they should be checked.

Mutex Support

The Inflexion UI Engine is not re-entrant, so access is serialized on multithreaded systems using mutexes. The porting layer implementations of the mutex functions should take appropriate action (such as suspending/resuming tasks) when two tasks require the same mutex at the same time.

Table 6-4. Mutex Support Function

Function	Description
IFXP_Mutex_Create IFXP_Mutex_Destroy	Create and destroy a mutex with the given name.
IFXP_Mutex_Lock	Gain control of a mutex if it is free. If not, suspend the caller for the requested time if appropriate.
IFXP_Mutex_Unlock	Release the mutex lock, allowing any waiting tasks to acquire the lock and resume if appropriate.

File System

The File API is used when `use_platform_files` is enabled. The Porting Layer allows the Inflexion UI engine to access files on a platform-specific file system.

The Inflexion UI engine uses drive letters to access package files. The Porting Layer maps these drive letters to platform specific locations (with or without drive letters).

Table 6-5. File System API

Function	Description
IFXP_File_Open IFXP_File_Close	Opens and closes a file handle. The remaining functions use the handle returned by open to access the file.
IFXP_File_Read	Reads data from an open file.
IFXP_File_Seek	Moves the file pointer to a specific location within the file.
IFXP_File_Find_First IFXP_File_Find_Next	Searches for the first and next occurrences of a filename containing the specified text.
IFXP_File_Find_Close	Cleans up any memory allocated by the file find functions.
IFXP_File_Size	Returns the size of the specified file.

Benchmarking

The function `IFXP_Benchmarking_Signal` is called whenever the Inflexion UI Engine emits a benchmarking signal. The signal will be one of the bitflags `IFXP_BENCHMARKING_*` defined in `nucleus/os/ui/inflexionui/engine/ifxui_porting.h`, and the implementation of `IFXP_Benchmarking_Signal` should take appropriate action to enable benchmarking on the system.

Custom Image Format Support

By default, the porting layer is configured to support unpacking of `.dds` and `.png` format image data. The following define can be enabled in `nucleus/os/ui/inflexionui/porting/inc/ifxui_porting_cfg.h` to manually configure graphic format support:

- `IFXP_JPEG_SUPPORT` - defines support for `.jpg/.jpeg` file unpacking. This support is disabled by default
- `IFXP_PNG_SUPPORT` - defines support for `.png` file unpacking. This support is enabled by default. defines support for `.png` file unpacking. Enabling this would require adding the “libpng” and “zlib” open source products to the system, or re-implementation of the source code in `nucleus/os/ui/inflexionui/porting/src/ifxui_porting_image_png.c`.

In this chapter:

SimpleEdit Module	75
SimpleEdit Module Definition File	75
SimpleEdit Module Header File	76
Module Implementation	76
Battery Module.	77
Module Header	77
Battery Module Implementation	78
Demo Module	78
Demo Module Definition File	78
Demo Module Header	79
Demo Module Implementation	79
Plugin Module	79
Plugin Module Definition File	79
Plugin Module Header	80
Plugin Module Implementation	80

SimpleEdit Module

The SimpleEdit module provides a mechanism by which text fields may be edited at runtime using the numeric keypad on a handheld device (the numbers 0-9, the * key, and the # key). It achieves this by offering two module services; firstly it offers event-handling services to the theme writer (thus can be set as the first port of call when handling UI events), and secondly it provides field value support services to other modules, storing field values and offering the Integration layer the value of the fields when needed.

SimpleEdit Module Definition File

The module definition file for the battery module is *nucleus/os/ui/inflexionui/modules/simpleedit/ifaxui_simpleedit.module*. The module definition file contains a single element definition:

```
<element type="eventHandler" linkPrefix="simpleEdit" bodyType="string"
supportFields="true">
  <param name="style" type="string"/>
  <param name="maxLength" type="int" defaultValue="10"/>
  <param name="inputMode" type="string"/>
  <param name="caretPosition" type="int" defaultValue="-1"/>
```

```
</element>
```

It indicates that the element is an event handler (for example, the element has no on-screen graphical presence), and it is capable of supplying the field data on behalf of other modules. The element creation link uses a string as a link body (actually the name of the field that is to be supported) and a set of parameters that indicates how the module will use the field data. The **style** parameter can be “password” (in which case the entered text appears as ‘*’s on screen), “text”, “phone”, or “numeric”.

In addition to the element definition, the module defines a global field, which indicates the editing mode the module is in (“auto-capitalize the first letter of each word”, “auto-capitalize the first letter of the first word of each sentence”, “always lower case”, “always upper case”, “numeric values of the keys”), and a link action to cycle the input modes through the five supported cases.

To use the event handler, it must be attached to the resource data element in a class definition in a theme template. For example, the following:

```
<text class="caption" halign="left" height="16" tappable="true"
valign="center">
<font face="{font16}" />
<resource data="" eventHandler="simpleEdit://caption?maxLength=12" />
</text>
```

attaches an instance of the `simpleedit` event handler element to the text class, “caption”. Note that the value of the **eventHandler** attribute is the creation link string. In this case, the field to be supported is the same as the class name of the element (so that the field is to be used as the resource for the element—if the element was a graphic type element, the field would have to expand to the path and filename for a suitable image).

SimpleEdit Module Header File

The module header can be found in the *nucleus/os/ui/inflexionui/modules/simpleedit* folder, entitled *ifxui_simpleedit.h*.

This file is automatically created by the UIIntegrator tool, and contains the prototypes for module initialization/de-initialization, a set of functions to support the event handler element, a set of generic field data retrieval functions and functions to return and cycle through the edit mode.

Module Implementation

Features to note in the implementation of the module include the implementation of `IFXM_ifxuisimpleedit_GetFieldStringSize_simpleEdit` (and the associated `GetFieldStringData`), where each field being handled by the module is passed back into the integration layer for use by the Engine—this will be called in preference over the equivalent functions in the module that normally offers the fields. Also note the

IFXM_ifxuisimpleedit_CreateElement_simpleEdit implementation—as part of this the module calls IFXI_BufferCreateString, passing in the name of the field that should be shared.

IFXI_BufferCreateString is created by the UIIntegrator tool, which knows the set of fields that can be shared this way from the module definition files (each shareable field has an editType=“string” attribute). The module that owns the field must create a suitably-sized buffer and fill it with the relevant field value (up to the max string length limit).

The final point of interest in this module is the IFXM_ifxuisimpleedit_ProcessElementKeyDownEvent_simpleEdit method, where the key press is directed first if an element with a simpleEdit resource handler has the UI focus. The key press is handled in the multitap part of the simpleedit module, which will decide whether or not to consume the key press (if not, the Runtime component passes on the key press to the theme, which can then take some other action) and whether the Runtime component needs to be updated (in which case IFXI_RequestFieldRefresh is called).

Battery Module

The module definition file for the battery module is in the folder *demo/inflexionui_default/modules/battery*, called *ifxui_battery.module*.

The module definition file contains two global fields (in other words, fields that have no menu item context):

```
<field name="batteryLevel" mode="output" dataType="int" minValue="0"
maxValue="100" />
<field name="batteryCharging" mode="output" dataType="int" minValue="0"
maxValue="1" />
```

The name attribute is how the fields are referenced in the UI theme. Both fields are of mode “output”, that indicates the engine should not allow you to modify them through the UI (in practical terms: these fields cannot be attached to a drag region, or used in a “scroll” action).

Both have type “int”, and both have a maximum and minimum value defined, to assist with using these fields as “displacement” fields in the theme design (see the *Mentor Embedded Inflexion UI Express User’s Guide* for more information on displacements).

Module Header

The module header can be found in the folder *demo/inflexionui_default/modules/battery*, entitled *ifxui_battery.h*. This is automatically created by the UIIntegrator tool, and contains the prototypes for module initialization/de-initialization along with two functions to retrieve the value of the two fields declared in the module definition file.

Battery Module Implementation

The module merely sets the `batteryLevel` field to “50” and the `batteryCharging` field to “1” (for example, battery charging). To extend this to dynamically changing values, a task would need adding to the module that changed the field values and notified Inflexion UI Runtime component via a call to `IFXI_RequestFieldRefresh`.

Demo Module

The demo module demonstrates how to define and use a module (run-time populated) menu, so the module controls the contents of the menu and has influence on how the menu data is presented. It also makes use of the `SimpleEdit` module that enables you to modify the menu item captions through a keypad.

Demo Module Definition File

The module definition file for the demo module is in the folder *demo/inflexionui_default/modules/demo*, called *ifxui_demo.module*.

The module definition file contains a set of link type definitions, a set of global fields and a menu definition. For the module menu case, the interesting components are:

```
<link linkPrefix="menuPlugin" bodyType="string" linkType="menu"
menu="edit" />
```

This defines a new link type for use in the theme (`menuPlugin://`), indicating that it can accept a link body of type text (so `menuPlugin://<some text here>`) and that it is a link that will attempt to open a menu, in particular the menu class called “edit” defined in this part:

```
<menu name="edit">
  <fields>
    <field name="icon" previewValue="{item1}" dataType="string"/>
    <field name="caption" previewValue="Unlock" editType="string"
dataType="string"/>
  </fields>
</menu>
```

This menu supports two “item scope” fields (so the field evaluate to different values depending on which item is referred to), both of type “string”. The preview values will be the ones displayed in the Inflexion UI Express if this menu is attached to a template that uses item classes called “icon” and “caption”. Finally, note that the “caption” field is declared as editable—this will alert the module to the fact that some other component (in this case `SimpleEdit`) may want to modify the contents of the field.

There are a large number of global fields used to store the state of a number of different sample menus in the template demo theme.

Demo Module Header

The module header can be found in the folder *demo/inflexionui_default/modules/demo*, entitled *ifxui_demo.h*. This is automatically created by the UIIntegrator tool, and contains the prototypes for module initialization/de-initialization along with a set of functions for opening and managing the module menu, a set of functions for retrieving the value of the field tokens supported by the modules, some link action methods and a set of methods for sharing the buffers used to store the “caption” field token strings between modules.

Demo Module Implementation

The module has an array of 8 menu items that is populated when the module is opened. The engine will follow the sequence outlined in [Figure 6-1](#), with the module returning the requested information (apart from in the case of the “caption” field, which will be served from the SimpleEdit module function IFXM_ifxuisimpleedit_GetFieldStringSize_simpleEdit). Eventually, the integrated menu will be displayed.

Note that the sample integration menu theme has attached “SimpleEdit” event handlers to the caption element (the resource element of the caption class has the **eventHandler** attribute set to the SimpleEdit creation link URI, passing in the name of the field to be handled in the URI body). Also notice that in each “slot active” state in the template, the “caption” element has been given the focus. The effect of this is to direct all user input events first to the SimpleEdit module—in particular numberpad keypresses will be absorbed by the SimpleEdit module and interpreted as edit commands on the caption field string contents. Changes to the contents of the “caption” element will be notified by calls to IFXI_RequestFieldRefresh.

Plugin Module

The plugin module demonstrates how to define and use a plugin user interface element, allowing a native application to render to part of the Inflexion UI scene. In this case, use a native “buffered” plugin element, so the module is writing to an Inflexion UI buffer, and the Inflexion UI Runtime component is rendering this to screen as if it was just another graphical element.

Plugin Module Definition File

The module definition file for the plugin module is in the folder *demo/inflexionui_default/modules/plugin*, called *ifxui_plugin.module*.

The module definition file contains a single element definition:

```
<element linkPrefix="demoPluginElement" type="graphic" bodyType="string"
defaultBody="" />
```

and a single link type definition used to control the element:

```
<link linkPrefix="ButtonLink" bodyType="int" defaultBody="1"/>
```

The first definition identifies the element as a plugin element (as the type is “graphic”, so it will render to screen) and that it is created with the special link URI “demoPluginElement://”. The second definition gives a link type by which to change the element drawing style – **call**

```
'ButtonLink://1' to draw a sine wave in the element, 'ButtonLink://2' to  
draw a cosine wave.
```

Plugin Module Header

The module header can be found in the folder *demo/inflexionui_default/modules/plugin*, entitled *ifxui_plugin.h*. This is automatically created by the UIIntegrator tool, and contains the prototypes for module initialization/de-initialization along with a set of functions to create and manage the plugin element and a function for the single link type declared in the module definition file.

Plugin Module Implementation

The module creates a “buffer mode” element in response to IFXM_ifxuiplugin_CreateElement_demoPluginElement being called (note that the only supported mode declared is IFX_MODE_ELEMENT_BUFFERED). The buffer details are communicated to the module in the call to IFXM_ifxuiplugin_ChangeElementMode_demoPluginElement where it is stored, and used in the function DrawWave. The module calls IFXI_RequestRefreshBufferedElement periodically to request that the Inflexion UI Runtime component redraw the element.

Chapter 8

Engine API Reference

This chapter describes the Engine API (IFXE) and some of the Inflexion UI Runtime functions.

In this chapter:

Engine API (IFXE)	81
Startup/Shutdown	82
Framework	90
Input	100
File Access	107

Engine API (IFXE)

The Engine API (IFXE) is the main application interface for Inflexion UI engine. The Inflexion UI Engine API prototypes are located in the header file *os/include/ui/ifxui_engine.h*, which should be included in your code as *ui/ifxui_engine.h*.

Types

All type definitions specific to the Inflexion UI Engine API are in a single header:
nucleus/os/include/ui/ifxui_defs.h.

List of Functions

This topic lists all function calls available to communicate with the UI Runtime engine, providing an alphabetized jump list.

- [IFXE_File_Close](#)
- [IFXE_File_Open](#)
- [IFXE_File_Read](#)
- [IFXE_File_Seek](#)
- [IFXE_File_Size](#)
- [IFXE_Get_Configuration](#)
- [IFXE_Initialize](#)

- [IFXE_Key_Down](#)
- [IFXE_Key_Up](#)
- [IFXE_Process_Timers](#)
- [IFXE_Refresh_All](#)
- [IFXE_Refresh_Display](#)
- [IFXI_RequestResourceSearch](#)
- [IFXE_Resume](#)
- [IFXE_Resume_Static_Animations](#)
- [IFXE_Save_State](#)
- [IFXE_Set_Configuration](#)
- [IFXE_Set_Idle](#)
- [IFXE_Shutdown](#)
- [IFXE_Suspend](#)
- [IFXE_Touch_Down](#)
- [IFXE_Touch_Move](#)
- [IFXE_Touch_Up](#)

Startup/Shutdown

The following functions are Inflexion UI Engine APIs related to the initialization and shutdown of the Inflexion UI engine.

- [IFXE_Initialize](#)
- [IFXE_Shutdown](#)
- [IFXE_Save_State](#)
- [IFXE_Suspend](#)
- [IFXE_Resume](#)

IFXE_Initialize

This function initializes the Inflexion UI Runtime using the supplied configuration.

Usage

```
IFX_RETURN_STATUS IFXE_Initialize (IFX_CONFIGURATION* config);
```

Parameters

- config

Structure containing the following parameters:

language

String containing the language (as defined in the platform definition file) to be used

left

Left offset of the menu on screen

top

Top offset of the menu on screen

width

Width of the desired output region in which the theme will be displayed

height

Height of the desired output region in which the theme will be displayed

display_mode

The display mode (as defined in the platform definition file) to be used

package_name

The name of a theme/package to be used

entrypoint_id

The entry point identifier selecting the initial page to be loaded in the Inflexion UI theme

restoreState

If set to 1, attempt to use stored data to load the UI into a particular state

Return Values

- IFX_SUCCESS
The command was successful.
- IFX_ERROR
Any error

Description

The IFXE_CONFIGURATION structure populated by this function is defined as follows:

```
typedef struct tagIFXE_CONFIGURATION
{
    char language[IFX_MAX_CONFIG_STRING_LENGTH];
    IFX_INT32 left;
    IFX_INT32 top;
    IFX_INT32 width;
    IFX_INT32 height;
    char display_mode[IFX_MAX_CONFIG_STRING_LENGTH];
    char package_name[IFX_MAX_CONFIG_STRING_LENGTH];
    char entrypoint_id[IFX_MAX_CONFIG_STRING_LENGTH];
    char restoreState;
} IFXE_CONFIGURATION;
```

If the configuration structure pointer is null, IFXE_Initialize will return IFX_ERROR and the Inflexion UI Runtime will not start.

The special value IFX_CENTER_DISPLAY should be used for the “left” or “top” members of the configuration structure when the output region should be centered in the middle of the display device.

If the display mode is missing (or not supported by the theme) Inflexion UI Runtime chooses the most appropriate display mode based on the width and height parameters of the configuration structure. This will be a mode that fits the screen without cropping the output image (so depending on aspect ratio of theme and output region, black borders are possible).

If display_mode is empty or invalid and the width and height parameters are less than or equal to 0, then the mode used will either be the default mode as specified for that device in the device definition file or, if that doesn’t exist, the first mode in the list of supported modes in the device definition file will be used.

If the display mode is empty, or if it refers to a mode that is not supported by the theme on the device, Inflexion UI Runtime will populate the display_mode parameter with the mode being used. Likewise, if the width and height parameters are not set, or if they are incorrect, Inflexion UI Runtime will fill these in with the appropriate values for the theme mode being used.

The entrypoint id identifies the initial UI page to load, and must be one of the entry point ids enumerated in the application definition file. The Inflexion UI theme will have a mapping between this id and the corresponding UI page. If this member of the structure is empty, the Inflexion UI Runtime component is initialized but no theme is loaded. The porting layer and integration layers are also initialized, but no canvases or other RAM resources are loaded. The UI Runtime component then awaits an additional call to IFXE_Initialize or IFXE_Set_Configuration.

The restoreState member should be set to 1 if a previously stored state (via a call to IFXE_Save_State) should be used to restore the UI to a particular state. If no state has been saved, or if state saving is not supported, this member is ignored.

Example

```
IFXE_CONFIGURATION engine_config;
```

```
lc_strcpy(engine_config.language, "en");

engine_config.left = 0;

engine_config.top = 0;

lc_strcpy(engine_config.display_mode, "VGA_P");

if ((screenWidth == 240) && (screenHeight == 320))
{
    lc_strncpy(engine_config.display_mode, "QVGA_P",
    IFX_MAX_CONFIG_STRING_LENGTH);
}

IFXE_Initialize(&engine_config);
```

Related Topics

[IFXE_Save_State](#)

[IFXE_Set_Configuration](#)

[IFXE_Shutdown](#)

IFXE_Shutdown

This function closes down Inflexion UI Runtime, de-initializing the porting layer and integration layer. Calls to IFXE methods after this has been called will fail.

Usage

```
IFX_RETURN_STATUS IFXE_Shutdown (void);
```

Return Values

- **IFX_SUCCESS**
The command was successful.
- **IFX_ERROR**
Any error

Related Topics

[IFXE_Initialize](#)

IFXE_Save_State

This function serializes the current state of the Inflexion UI pages and writes the saved data to some (platform-specific) location, ready to restore the UI to the current state at some later time. Note that this function can take some time to complete, thus there is an “abort save” mechanism provided to interrupt this method and clean up any saved data in the case where the framework suspends and resumes Inflexion UI with a small interval.

Usage

```
IFX_RETURN_STATUS IFXE_Save_State (int *abortSave);
```

Parameters

- **abortSave**
Address of an integer that is used as a “signal” if the serialization process should be terminated prematurely. If this parameter is NULL, or if the value of the memory pointed to is 0, then the serialization process will continue. If the value of the memory that abortSave points to changes to a non-zero value, serialization will be interrupted.

Return Values

- **IFX_SUCCESS**
The command was successful.
- **IFX_ERROR**
State could not be saved, or state saving not supported.

Related Topics

[IFXE_Resume](#)

[IFXE_Initialize](#)

IFXE_Suspend

This function puts Inflexion UI Runtime into an idle state (no animations) and releases the foreground and background canvas memory.

After calling this function, valid IFXE calls are IFXE_Resume (resume existing UI theme), IFXE_Initialize (launch new UI theme) and IFXE_Shutdown.

Usage

```
IFX_RETURN_STATUS IFXE_Suspend (void);
```

Return Values

- **IFX_SUCCESS**
The command was successful.
- **IFX_ERROR_ENGINE_NOT_READY**
The runtime is in the process of restarting following a critical error.
- **IFX_ERROR**
Any other error.

Related Topics

[IFXE_Resume](#)

[IFXE_Initialize](#)

[IFXE_Shutdown](#)

IFXE_Resume

This function restores Inflexion UI Runtime from the suspended state. This reallocates the foreground and background canvasses and repaints the display.

Usage

```
IFX_RETURN_STATUS IFXE_Resume (void);
```

Return Values

- **IFX_SUCCESS**
The command was successful.
- **IFX_ERROR_ENGINE_NOT_READY**
The runtime is in the process of restarting following a critical error.
- **IFX_ERROR**
Any other error.

Related Topics

[IFXE_Suspend](#)

[IFXE_Initialize](#)

Framework

The following functions are Runtime APIs related to the configuration of the Inflexion UI engine.

- [IFXE_Process_Timers](#)
- [IFXE_Set_Configuration](#)
- [IFXE_Get_Configuration](#)
- [IFXE_Refresh_All](#)
- [IFXE_Refresh_Display](#)
- [IFXE_Resume_Static_Animations](#)
- [IFXE_Set_Idle](#)

IFXE_Process_Timers

This function drives the Inflexion UI Runtime's internal state machine.

Calls should be called as a result of a timer expiration event following calls to IFXP_Timer_Schedule. If a timer is not available, this function can be called periodically at a period no longer than 40ms.

Not all calls to this function will cause a frame to be drawn; however all frames will be drawn in the context of a call to this function. The time stamp passed to this function must always be greater than the previous value and should represent a real time source in milliseconds. Calling this function with a time less than or equal to the previous value has no effect.

Usage

```
IFX_RETURN_STATUS IFXE_Process_Timers (IFX_UINT32 time_upper, IFX_UINT32  
    time_lower);
```

Parameters

- **time_upper**
Time stamp in ms to process. This should be the upper DWORD of the 64-bit current system time in ms.
- **time_lower**
Time stamp in ms to process. This should be the lower DWORD of the 64-bit current system time in ms.

Return Values

- **IFX_SUCCESS**
The command was successful.
- **IFX_ERROR**
Any error

Example

```
struct timeval tp;  
IFX_UINT32 time_upper, time_lower;  
  
gettimeofday(&tp, NULL);  
  
time_upper = (tp.tv_sec / (UINT_MAX/1000));  
time_lower = (tp.tv_sec % (UINT_MAX/1000)) * 1000;  
time_lower += tp.tv_usec / 1000;  
  
IFXE_Process_Timers(time_upper, time_lower);
```

Related Topics

[IFXE_Resume](#)

[IFXE_Set_Idle](#)

[IFXE_Suspend](#)

IFXE_Set_Configuration

This function configures Inflexion UI Runtime with the specified language, display mode, package name, or entry point. All parameters are optional. Where a configuration option is not required, it is populated with an empty string (“”) or 0 as appropriate.

Calling this function with a valid configuration different to the current one causes the current Inflexion UI theme to reset to the root page when the new configuration is applied on the next call to IFXE_Process_Timers. The exception to this rule is if the configuration differs only in the entrypoint_id, in which case the corresponding new page is opened without returning the theme to the root page.

Calling this function with a configuration that matches the existing configurations has no effect.

Usage

```
IFX_RETURN_STATUS IFXE_Set_Configuration (IFX_CONFIGURATION* config);
```

Parameters

- config

Structure containing the following parameters:

language

String containing the language (as defined in the platform definition file) to be used

left

Left offset of the menu on screen

top

Top offset of the menu on screen

display_mode

The display mode (as defined in the platform definition file) to be used

package_name

The name of a theme/package to be used

entrypoint_id

The entry point identifier selecting the page to be loaded in the Inflexion UI theme

Return Values

- IFX_SUCCESS

The command was successful.

- IFX_ERROR_ASYNC_BLOCKING

The Inflexion UI Runtime is in the asynchronous link launch state and unable to accept input.

- **IFX_ERROR_ENGINE_NOT_READY**
The runtime is in the process of restarting following a critical error.
- **IFX_ERROR**
Any other error

Example

See the IFXE_Initialize example.

Related Topics

[IFXE_Initialize](#)

[IFXE_Get_Configuration](#)

IFXE_Get_Configuration

This function populates the configuration structure with the current display, theme, and language configuration as specified during IFXE_Set_Configuration.

Usage

```
IFX_RETURN_STATUS IFXE_Get_Configuration (IFX_CONFIGURATION* config);
```

Parameters

- **config**
Pointer to configuration structure to be populated by this function

Return Values

- **IFX_SUCCESS**
The command was successful.
- **IFX_ERROR_ASYNC_BLOCKING**
The Inflexion UI Runtime is in the asynchronous link launch state and unable to accept input.
- **IFX_ERROR**
Any other error

Related Topics

[IFXE_Initialize](#)

[IFXE_Set_Configuration](#)

IFXE_Refresh_All

This function causes the Inflexion UI Runtime to refresh all module field data and update the whole display.

Usage

```
IFX_RETURN_STATUS IFXE_Refresh_All (void);
```

Return Values

- **IFX_SUCCESS**
The command was successful.
- **IFX_ERROR_ASYNC_BLOCKING**
The Inflexion UI Runtime is in the asynchronous link launch state and unable to accept input.
- **IFX_ERROR_ENGINE_NOT_READY**
The runtime is in the process of restarting following a critical error.
- **IFX_ERROR**
Any other error.

Related Topics

[IFXE_Refresh_Display](#)

IFXE_Refresh_Display

This function causes Inflexion UI Runtime to refresh the whole display area with the current frame regardless of the areas that have changed.

Usage

```
IFX_RETURN_STATUS IFXE_Refresh_Display (void);
```

Return Values

- **IFX_SUCCESS**
The command was successful.
- **IFX_ERROR_ENGINE_NOT_READY**
The runtime is in the process of restarting following a critical error.
- **IFX_ERROR**
Any other error.

Related Topics

[IFXE_Refresh_All](#)

IFXE_Resume_Static_Animations

This function causes any terminated static animations in currently active layouts to be restarted. Typically, this would be used following a call to [IFXE_Set_Idle](#) (which would terminate any static animations) if the application required the statics to be restarted.

Usage

```
IFX_RETURN_STATUS IFXE_Resume_Static_Animations (void);
```

Return Values

- **IFX_ERROR_SUCCESS**
Static animations restarted if necessary
- **IFX_ERROR_ASYNC_BLOCKING**
Runtime is in the asynchronous link state
- **IFX_ERROR_ENGINE_NOT_READY**
Runtime is restarting after a serious error
- **IFX_ERROR**
Any other error

Related Topics

[IFXE_Suspend](#)

[IFXE_Resume](#)

[IFXE_Refresh_All](#)

[IFXE_Refresh_Display](#)

IFXE_Set_Idle

This function causes Inflexion UI Runtime to go into an idle state. All animations are brought to their end positions and no new animations are scheduled.

Usage

```
IFX_RETURN_STATUS IFXE_Set_Idle (void);
```

Return Values

- **IFX_SUCCESS**
The command was successful.
- **IFX_ERROR_ASYNC_BLOCKING**
The Inflexion UI Runtime is in the asynchronous link launch state and unable to accept input.
- **IFX_ERROR_ENGINE_NOT_READY**
The runtime is in the process of restarting following a critical error.
- **IFX_ERROR**
Any other error.

Related Topics

[IFXE_Async_Link_Complete](#)

[IFXE_Resume](#)

[IFXE_Suspend](#)

[IFXE_Shutdown](#)

Input

The following functions are Engine APIs related to how Inflexion UI Runtime handles user input, such as keypresses and “pointer” events.

- [IFXE_Key_Down](#)
- [IFXE_Key_Up](#)
- [IFXE_Touch_Up](#)
- [IFXE_Touch_Down](#)
- [IFXE_Touch_Move](#)

IFXE_Key_Down

This function routes the key event code to the Inflexion UI Runtime.

Only one key can be down at any one time. For example, a call to `key_down(code1)` can only be followed by a call to `key_up(code1)`, not `key_down(code2)`. Actions are applied in the Inflexion UI Runtime on the key down event.

Usage

```
IFX_RETURN_STATUS IFXE_Key_Down (IFX_INT32 code);
```

Parameters

- code

Key code. The standard key codes are defined in *ifxui_defs.h* as the `IFX_KEY_CODE` enumeration:

```
typedef enum {  
    IFX_KEY_CODE_HASH = 0x0000FFFF,  
    IFX_KEY_CODE_asterisk,  
    IFX_KEY_CODE_UP,  
    IFX_KEY_CODE_DOWN,  
    IFX_KEY_CODE_LEFT,  
    IFX_KEY_CODE_RIGHT,  
    IFX_KEY_CODE_SELECT,  
    IFX_KEY_CODE_BACK  
} IFX_KEY_CODE;
```

Additional codes may be added using the device definition file.

Return Values

- IFX_SUCCESS
The command was successful.
- IFX_ERROR_ASYNC_BLOCKING
The Inflexion UI Runtime is in the asynchronous link launch state and unable to accept input.
- IFX_ERROR_ENGINE_NOT_READY
The runtime is in the process of restarting following a critical error.
- IFX_ERROR
Any other error

Example

```
IFXE_Key_Down (IFX_KEY_CODE_UP);
```

Related Topics

[IFXE_Key_Up](#)

[IFXE_Touch_Up](#)

[IFXE_Touch_Down](#)

[IFXE_Touch_Move](#)

IFXE_Key_Up

This function routes the key event code to the Inflexion UI Runtime.

Usage

```
IFX_RETURN_STATUS IFXE_Key_Up (IFX_INT32 code);
```

Parameters

- code
Key code (see [IFXE_Key_Down](#) for codes).

Return Values

- IFX_SUCCESS
The command was successful.
- IFX_ERROR_ASYNC_BLOCKING
The Inflexion UI Runtime is in the asynchronous link launch state and unable to accept input.
- IFX_ERROR_ENGINE_NOT_READY
The runtime is in the process of restarting following a critical error.
- IFX_ERROR
Any other error

Example

```
IFXE_Key_Up (IFX_KEY_CODE_UP);
```

Related Topics

[IFXE_Key_Down](#)

[IFXE_Touch_Down](#)

[IFXE_Touch_Up](#)

[IFXE_Touch_Move](#)

IFXE_Touch_Up

This function routes the touch panel event at location (x,y) to the Inflexion UI Runtime. The coordinates are in the global screen space. Actions are applied in the Inflexion UI Runtime on the touch up event (other than dragging events, which apply on the touch move event).

Usage

```
IFX_RETURN_STATUS IFXE_Touch_Up (IFX_UINT32 touch_id, IFX_INT32 x,  
                                IFX_INT32 y);
```

Parameters

- touch_id
Touch id – always zero
- x
Touch x location in screen coordinates
- y
Touch y location in screen coordinates

Return Values

- IFX_SUCCESS
The command was successful.
- IFX_ERROR_ASYNC_BLOCKING
The Inflexion UI Runtime is in the asynchronous link launch state and unable to accept input.
- IFX_ERROR_ENGINE_NOT_READY
The runtime is in the process of restarting following a critical error.
- IFX_ERROR
Any other error

Example

```
IFXE_Touch_Up(0, x, y);
```

Related Topics

[IFXE_Key_Down](#)

[IFXE_Touch_Down](#)

[IFXE_Key_Up](#)

[IFXE_Touch_Move](#)

IFXE_Touch_Down

This function routes the touch panel event at location (x,y) to the Inflexion UI Runtime. The coordinates are in the global screen space.

Usage

```
IFX_RETURN_STATUS IFXE_Touch_Down (IFX_UINT32 touch_id, IFX_INT32 x,  
    IFX_INT32 y);
```

Parameters

- touch_id
Touch ID – always zero
- x
Touch x location in screen coordinates
- y
Touch y location in screen coordinates

Return Values

- IFX_SUCCESS
The command was successful.
- IFX_ERROR_ASYNC_BLOCKING
The Runtime is in the asynchronous link launch state and unable to accept input.
- IFX_ERROR_ENGINE_NOT_READY
The runtime is in the process of restarting following a critical error.
- IFX_ERROR
Any other error

Example

```
IFXE_Touch_Down(0, x, y);
```

Related Topics

[IFXE_Key_Down](#)

[IFXE_Key_Up](#)

[IFXE_Touch_Up](#)

[IFXE_Touch_Move](#)

IFXE_Touch_Move

This function routes the touch panel event at location (x,y) to the Inflexion UI Runtime. This function should be called each time the position of the input device moves. The frequency of updates affects the smoothness of the response. Therefore, it should be every 40ms or less.

Ideally, this function is called after a touch down and before a touch up event; however, the Inflexion UI Runtime is robust if this is not the case.

Usage

```
IFX_RETURN_STATUS IFXE_Touch_Move (IFX_UINT32 touch_id, IFX_INT32 x,  
    IFX_INT32 y);
```

Parameters

- touch_id
Touch id – always zero
- x
Touch x location in screen coordinates
- y
Touch y location in screen coordinates

Return Values

- IFX_SUCCESS
The command was successful.
- IFX_ERROR_ASYNC_BLOCKING
The Inflexion UI Runtime is in the asynchronous link launch state and unable to accept input.
- IFX_ERROR_ENGINE_NOT_READY
The runtime is in the process of restarting following a critical error.
- IFX_ERROR
Any other error

Example

```
IFXE_Touch_Move(0, x, y);
```

Related Topics

[IFXE_Key_Down](#)

[IFXE_Key_Up](#)

[IFXE_Touch_Up](#)

[IFXE_Touch_Down](#)

File Access

The following functions are Engine APIs providing access to theme resources, including files contained within a “ROMized” theme.

- [IFXE_File_Open](#)
- [IFXE_File_Read](#)
- [IFXE_File_Seek](#)
- [IFXE_File_Size](#)
- [IFXE_File_Close](#)
- [IFXI_RequestResourceSearch](#)

IFXE_File_Open

This function opens the named file, even if the file is inside an Inflexion UI ROMized theme. The string passed into the function must be the full path (including drive letter). The Porting Layer is passed this information by the Engine component, and any modules that require use of this feature must make use of the method *IFXI_RequestResourceSearch* to acquire this information.

Usage

```
IFX_RETURN_STATUS IFXE_File_Open(IFXE_FILE* handle, const char* file_name);
```

Parameters

- **handle**
Pointer to an *IFXE_FILE* object to hold the file handle on success.
- **file_name**
Full path and name of the file to open. Limited in length to *IFX_MAX_FILE_NAME_LEN*.

Return Values

- **IFX_SUCCESS**
The file is open.
- **IFX_ERROR**
Any error

Related Topics

[IFXE_File_Read](#)

[IFXE_File_Seek](#)

[IFXE_File_Close](#)

IFXE_File_Read

Reads up to “size” bytes from the open file and places them in the buffer pointed to by the `addr` parameter. The actual number of bytes read is stored in the variable pointed to by `bytes_read`—this is less than “size” if an “end of file” is detected during the read.

Usage

```
IFX_RETURN_STATUS IFXE_File_Read(IFXE_FILE handle, void* addr, IFX_UINT32
    size, IFX_UINT32* bytes_read);
```

Parameters

- `handle`
The `IFXE_FILE` object value set by the call to `IFXE_File_Open` for this file.
- `addr`
Pointer to the data buffer to be used to hold the file data.
- `size`
Size of the data buffer specified by the `addr` parameter.
- `bytes_read`
Pointer to an `IFX_UINT32` variable to hold the actual number of bytes placed into the `addr` parameter.

Return Values

- `IFX_SUCCESS`
The file was read successfully.
- `IFX_ERROR`
Any error

Related Topics

[IFXE_File_Open](#)

[IFXE_File_Seek](#)

[IFXE_File_Size](#)

IFXE_File_Seek

Move the current file read location. The starting point for the seek operation is the start of the file, the current read position, or the end of the file. The seek operation can be performed in either direction.

Usage

```
IFX_RETURN_STATUS IFXE_File_Read(IFXE_FILE handle, IFX_INT32 offset, IFX_SEEK  
    seek_mode) ;
```

Parameters

- **handle**
The IFXE_FILE object value set by the call to IFXE_File_Open for this file.
- **offset**
Moves the read position the corresponding number of bytes forward (for positive values of offset) or backwards (for negative values of offset) from the position defined by the seek_mode parameter.
- **seek_mode**
Sets the starting point for the seek operation:
 - IFX_SEEK_SET starts from the beginning of the file.
 - IFX_SEEK_CUR starts from the current read position.
 - IFX_SEEK_END starts from the end of the file.

Return Values

- **IFX_SUCCESS**
The read point was moved successfully.
- **IFX_ERROR**
Any error

Related Topics

[IFXE_File_Open](#)

[IFXE_File_Read](#)

IFXE_File_Size

Queries the size of a file previously opened with IFXE_File_Open.

Usage

```
IFX_RETURN_STATUS IFXE_File_Size(IFXE_FILE handle, IFX_UINT32* size);
```

Parameters

- **handle**
The IFXE_FILE object value set by the call to IFXE_File_Open for this file.
- **size**
Pointer to an IFX_UINT32 variable to hold the size in bytes of the file.

Return Values

- **IFX_SUCCESS**
The operation was a success.
- **IFX_ERROR**
Any error

Related Topics

[IFXE_File_Open](#)

IFXE_File_Close

Closes a file previously opened with IFXE_File_Open.

Usage

```
IFX_RETURN_STATUS IFXE_File_Close(IFXE_FILE handle);
```

Parameter

- **handle**
The IFXE_FILE object value set by the call to IFXE_File_Open for this file.

Return Values

- **IFX_SUCCESS**
The file was closed successfully.
- **IFX_ERROR**
Any error

Related Topics

[IFXE_File_Open](#)

IFXI_RequestResourceSearch

This function can only be called from IFXM - ExecuteLink. It finds a specified file in the resource search path and returns a full file path. This function can succeed if another module has exclusivity.

Usage

```
IFX_RETURN_STATUS IFXI_RequestResourceSearch (  
    IFX_HMODULEID hModuleID,  
    IFX_HMENUID hMenuId,  
    const IFX_WCHAR* szResource,  
    IFX_WCHAR* szFullPath);
```

Parameters

- **hModuleId**
The handle supplied to the module by IFXM - Initialize.
- **hMenuId**
The menu identifier to use when the module returns to the Integration Layer regarding the supplied menu context. The Inflexion UI Runtime uses this to determine which search path to use.
- **szResource**
The filename to find, including extension. Wildcards are not allowed.
- **szFullPath**
The pointer to a buffer into which Inflexion UI places the file's absolute path. The maximum length is IFX_MAX_PATH_LENGTH characters and the buffer is large enough to hold this.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR**
File not found, or other general error.
- **IFX_ERROR_ASYNC_BLOCKING**
The Inflexion UI Runtime is currently executing an asynchronous link and cannot process the request at this time.
- **IFX_ERROR_ENGINE_NOT_READY**
The runtime is in the process of restarting following a critical error.

- String containing the file's absolute path.

Related Topics

[IFXI_RequestQueryActiveItem](#)

[IFXM - ExecuteLink](#)

Chapter 9

Module Integration API Reference

This chapter defines the functions implemented by modules. The integration layer uses these functions to perform module-supported actions. The following is the naming convention:

- “IFXI”- prefixed functions are calls into the Integration Layer from the modules.
- “IFXM”- prefixed functions are calls into the modules from the Integration Layer, and are implemented by the module author.

In this chapter:

C Language Modules	115
General Types	116
List of Methods	120
Module Framework	122
Menus	126
Field Data	149
Plugin	161
Functions	200
Exclusivity	206
Shared Buffers	210

C Language Modules

The prototypes for these functions are found in the header file *nucleus/os/include/ui/ifxui_module_integration.h*, apart from [IFXI_ReleaseExclusivity](#) and [IFXI_RequestExclusivity](#), which are prototyped on demand in the relevant module header file and implemented in the integration layer by the UIIntegrator tool.

General Types

All general type definitions common to all of the different Inflexion UI APIs are in a single header, *nucleus/os/include/ui/ifxui_defs.h*

Table 9-1. General Type Definitions

Type Definition	Description
IFX_BUFFER_FORMAT	Used when describing the properties of a plugin or integrated application element, this enum describes the format of the data buffer containing the image data that is shared between the module and the Inflexion runtime component. Currently supported formats are: <ul style="list-style-type: none">• IFX_32BPP_RGBA - Buffer is 32-bit-per-pixel, with each DWORD in the buffer containing the R, G, B, A components of the pixel, in that order. If the Alpha channel is unused, the A component should be set to 0xFF.• IFX_24BPP_RGB - Buffer is 24-bit-per-pixel, 3 bytes per pixel containing the R, G, B components of the pixel in that order.• IFX_16BPP_RGB565 - Buffer is 16 bit-per-pixel, 2 bytes per pixel (unsigned short) containing the R(5), G(6), B(5) components of the pixel in that order.
IFX_BUFFER_SIZE	Used when describing the properties of a plugin or integrated application element, this structure describes the dimensions of the the data buffer containing the image data that is shared between the module and the Inflexion runtime component.
IFX_BUFFERED_RENDER_CONTEXT	A structure used to describe the properties of a plugin or integrated application element data buffer containing image data to be shared between the module and the Inflexion runtime component. The module should update the pBuffer member to point to the buffer that satisfies the height, width and format members in the call to IFXM - ChangeElementMode .
IFX_CANVAS	This structure describes a rendering canvas for use in the porting layer when not using OpenGL. The porting layer must provide the runtime component with a pointer to one of these structures, containing a pointer to a buffer that the runtime component can write to, along with height, width, and stride information.

Table 9-1. General Type Definitions (cont.)

Type Definition	Description
IFX_DIRECT_RENDER_CONTEXT	Non-OpenGL rendering modes only, and only relevant to plugin/integrated application elements of type IFX_MODE_ELEMENT_BUFFERED_AND_DIRECT. A structure used to describe the properties of a plugin or integrated application element render context to be used by the module to update the representation of the element. The module should record the pDisplay pointer, and use this in the GDI calls when drawing the plugin element representation. The iClipHeight and iClipWidth should be used to constrain the viewport to avoid writing outside of the element's "client area".
IFX_DISPLAY	This structure contains the following information about a display: <ul style="list-style-type: none">• frame buffer• dimensions• offsets• rotation that should be applied• display color depth (bits per pixel)• number of samples for MSAA• an implementation-specific internal pointer
IFX_ELEMENT_MODE	This enum specifies in what way a plugin/integrated application element wishes to represent the image data. Options are: <ul style="list-style-type: none">• IFX_MODE_ELEMENT_OPENGL_TEXTURE - An OpenGL Texture is created and managed externally. Only supported if the Runtime engine is built in OpenGL mode• IFX_MODE_ELEMENT_BUFFERED_AND_DIRECT - The plugin supports a buffered mode (where the module provides the Runtime component with a buffer to draw from) and also a direct mode where it is responsible for writing to the display directly (via whatever GDI the porting layer is using to render the Inflexion canvas). Only supported if the Runtime engine is not using OpenGL• IFX_MODE_ELEMENT_BUFFERED - The plugin supports buffered mode data (where the module provides the runtime component with a buffer to draw from). This mode is supported for all runtime render modes.

Table 9-1. General Type Definitions (cont.)

Type Definition	Description
IFX_ELEMENT_PROPERTY	This structure is used in the call to IFXM - CreateElement to negotiate the properties of the plugin/integrated application element visual representation. The module must fill in the mode member to indicate what way the image data will be shared (see the definition of IFX_ELEMENT_MODE). The module should inspect the requiredBufferHeight and requiredBufferWidth members to see if the theme writer has given the module a hint as to what sort of buffer size would be most efficient. The module must update these members if they are not set, and may update them if the module has good reason to use a different buffer size. If the element contains any transparent or partially-transparent pixels, it should set the “translucency” parameter to “1”, otherwise the runtime will assume the element is opaque. If the render mode is “Direct OpenGL 2.0” and the element has transparent or partially-transparent pixels where the transparency is already factored into the color channels (“Pre-multiplied alpha”), the “hasPreMultipliedAlpha” member should be set to “1”. Finally, the module may change the value of requiredBufferFormat if it wishes to use a buffer format other than IFX_32BPP_RGBA.
IFX_KEY_CODE	Inflexion UI Runtime contains a set of abstracted key codes to cover the most common user actions. A device should map hardware keypresses to the relevant IFX_KEY_CODE_* value. Hardware scan codes smaller than 0xFFFF can be passed into Inflexion UI Runtime without being interpreted as an IFX_KEY_CODE key, but the UI theme must be specially tailored to respond to these events.
IFX_OPENGL_RENDER_CONTEXT	OpenGL rendering modes only, and only relevant to plugin/integrated application elements of type IFX_MODE_ELEMENT_OPENGL_TEXTURE. This structure defines the Open GL texture that contains the image data for the element. The module must fill in the texture member with the texture id, and may update the tex_width and tex_height members if it wishes some subsection of the texture to be rendered into the display area.

Table 9-1. General Type Definitions (cont.)

Type Definition	Description
IFX_POSITION	Non-OpenGL rendering modes only, and only relevant to plugin/integrated application elements of type IFX_MODE_ELEMENT_BUFFERED_AND_DIRECT. This structure is used to transfer position information to the module from the runtime component in calls to IFXM - PositionElement , but only while the element is in the “direct rendering” mode.
IFX_RETURN_STATUS	This is an enumeration. All function calls must return either IFX_SUCCESS or one of the error codes listed in the header file. Specific error codes rarely affect the response of the Inflexion UI Runtime to an error. Returning IFX_ERROR is usually sufficient if a request cannot be completed. The IFX_ERROR_USER_DEFINED error value gives the starting point for error codes defined and used by modules.
IFX_SORT_DIRECTION	This enumeration is used when opening module menus and indicates how the menu item sorting should be done. Currently supported values are: <ul style="list-style-type: none">• IFX_SORT_ASCENDING, depending on sort type, the order will have the items with the smallest criteria value at the top of the menu.• IFX_SORT_DESCENDING, depending on sort type, the order will have the items with the smallest criteria value at the bottom of the menu.
IFX_SORT_TYPE	This enumeration is used when opening module menus, and indicates how the menu item sorting should be performed. Currently supported values are: <ul style="list-style-type: none">• IFX_SORT_CASE_SENSITIVE - The field used for sorting the items will sort them in alphabetical order, differentiating between text case (for example, “a” and “A” would be different). The sort criteria is the difference in ASCII value of the first differing character in two values of the field.• IFX_SORT_CASE_INSENSITIVE - The field used for sorting the items will sort them in alphabetical order, ignoring differences in text case (for example, “a” and “A” would be the same). The sort criteria is the difference in ASCII value of the first differing character in two values of the field after they've been forced to be lower case.

Table 9-1. General Type Definitions (cont.)

Type Definition	Description
IFX_TIME	This is a platform-independent 32-bit time type. When time values are passed between the Integration Layer and modules, they should be this type. It is usually represented as the number of seconds since 1st January 1970. Note that the integration layer does not adjust the time values to be the local time. If this behavior is required, then the time values must be altered as necessary before being passed to the integration layer.
IFX_WCHAR	This is a platform-independent 16-bit or 32-bit character type. String values that are passed between Inflexion UI Runtime and the Integration Layer are always based on arrays of this type. The Inflexion UI Runtime uses the Unicode character set (UCS).

List of Methods

This topic lists all method calls available to communicate between modules and the integration layer, providing an alphabetized jump list.

- [IFXE_Async_Link_Complete](#)
- [IFXI_BufferCreateString](#)
- [IFXI_BufferDestroy](#)
- [IFXI_BufferSetFocus](#)
- [IFXI_BufferUnsetFocus](#)
- [IFXI_ReleaseExclusivity](#)
- [IFXI_RequestExclusivity](#)
- [IFXI_RequestExecuteLink](#)
- [IFXI_RequestFieldRefresh](#)
- [IFXI_RequestFullScreenStart](#)
- [IFXI_RequestFullScreenStop](#)
- [IFXI_RequestMenuRefresh](#)
- [IFXI_RequestQueryActiveItem](#)
- [IFXI_RequestRefreshBufferedElement](#)
- [IFXI_RequestResizeBufferedElementBuffer](#)

- [IFXI_RequestSetActiveItem](#)
- [IFXI_RequestTriggerKey](#)
- [IFXM - ActivateElement](#)
- [IFXM - BufferCreateString](#)
- [IFXM - BufferDestroy](#)
- [IFXM - BufferSetFocus](#)
- [IFXM - BufferUnsetFocus](#)
- [IFXM - ChangeElementMode](#)
- [IFXM - CloseMenu](#)
- [IFXM - CreateElement](#)
- [IFXM - DeactivateElement](#)
- [IFXM - DestroyElement](#)
- [IFXM - ExclusivityStatusChange](#)
- [IFXM - ExecuteLink](#)
- [IFXM - GetElementCaretPosition](#)
- [IFXM - GetFieldData](#)
- [IFXM - GetFieldStringSize](#)
- [IFXM - GetFirstActiveItem](#)
- [IFXM - GetItemCount](#)
- [IFXM - GetLoadOnDemand](#)
- [IFXM - GetMenuType](#)
- [IFXM - GetPlaceholderData](#)
- [IFXM - GetPlaceholderSize](#)
- [IFXM - GetSortDirection](#)
- [IFXM - GetSortFieldData](#)
- [IFXM - GetSortFieldSize](#)
- [IFXM - GetSortType](#)
- [IFXM - Initialize](#)
- [IFXM - OpenMenu](#)

- [IFXM - PaintElement](#)
- [IFXM - PositionElement](#)
- [IFXM - ProcessElementKeyDownEvent](#)
- [IFXM - ProcessElementKeyUpEvent](#)
- [IFXM - ProcessElementStylusCancelEvent](#)
- [IFXM - ProcessElementStylusDragEvent](#)
- [IFXM - ProcessElementStylusUpEvent](#)
- [IFXM - SetActiveItem](#)
- [IFXM - SetElementFocus](#)
- [IFXM - SetFieldData](#)
- [IFXM - Shutdown](#)
- [IFXM - UnsetElementFocus](#)

Module Framework

The following method templates initialize and deinitialize a module. It is possible for a module to record the module handles it creates or reference count the handle if it is reused.

- [IFXM - Initialize](#)
- [IFXM - Shutdown](#)

IFXM - Initialize

The Integration Layer calls this function when initialization is required. If a module returns an error code from Initialize, none of that module's functionality is available via the UI. Other modules in the system are initialized as normal.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_Initialize  
(IFX_HMODULEID hModuleId, IFX_HMODULE* phModule);
```

Parameters

- **hModuleId**
A handle the module uses to identify itself to the Integration Layer if it uses callback API functions.
- **phModule**
Out pointer to an IFX_HMODULE variable into which the module puts a handle value. This value is returned to the module on every future API call. The module implementer can use any value. The handle type is large enough to hold a pointer or an IFX_INT32. Typically, the handle is a pointer to a module's internal global data structure.

Return Values

- **IFX_SUCCESS**
Initialization completed successfully.
- **IFX_ERROR**
Any error. IFXM - Shutdown is not called, therefore, you must free any memory allocated before returning this error.

Example

```
IFX_RETURN_STATUS IFXM_ifxuisimpleedit_Initialize(  
    IFX_HMODULEID      hModuleId,  
    IFX_HMODULE        *phModule)  
{  
    SIMPLE_EDIT_SESSION* pLocalSession;  
  
    pLocalSession =  
    IFXE_AllocateMemoryUnsafe(sizeof(SIMPLE_EDIT_SESSION));  
    if (pLocalSession != NULL)  
    {  
        pLocalSession->hModID = hModuleId;  
        pLocalSession->lastKey = CONTROL_KEY;  
        pLocalSession->pCurrentElement = NULL;  
        *phModule = (IFX_HMODULE)pLocalSession;  
  
        return IFX_SUCCESS;  
    }  
}
```

```
        return IFX_ERROR;  
    }
```

Related Topics

[IFXM - Shutdown](#)

IFXM - Shutdown

The Integration Layer calls this function when final cleanup is required. If the module is initialized more than once, the module must reference count the active sessions and only perform final deinitialization when the final session shuts down.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_ShutDown (IFX_HMODULE hModule);
```

Parameters

- **hModule**
The module handle returned by IFXM - Initialize.

Return Values

- **IFX_SUCCESS**
Shutdown completed successfully.

Example

```
IFX_RETURN_STATUS IFXM_ifxuisimpleedit_ShutDown(  
    IFX_HMODULE      hModule)  
{  
    SIMPLE_EDIT_SESSION* pLocalSession = (SIMPLE_EDIT_SESSION*)hModule;  
  
    if (pLocalSession != NULL)  
    {  
        IFXE_FreeMemoryUnsafe(pLocalSession);  
    }  
  
    return IFX_SUCCESS;  
}
```

Related Topics

[IFXM - Initialize](#)

Menus

The following method templates are all related to supporting module menus.

- [IFXM - OpenMenu](#)
- [IFXM - CloseMenu](#)
- [IFXM - GetItemCount](#)
- [IFXM - GetMenuType](#)
- [IFXM - GetLoadOnDemand](#)
- [IFXM - GetPlaceholderSize](#)
- [IFXM - GetPlaceholderData](#)
- [IFXM - GetSortFieldSize](#)
- [IFXM - GetSortFieldData](#)
- [IFXM - GetSortDirection](#)
- [IFXM - GetSortType](#)
- [IFXM - GetFirstActiveItem](#)
- [IFXM - SetActiveItem](#)
- [IFXI_RequestMenuRefresh](#)
- [IFXI_RequestQueryActiveItem](#)
- [IFXI_RequestSetActiveItem](#)

IFXM - OpenMenu

The Integration Layer calls this function when you click a link that opens a menu of a type served by this module. The module prepares the necessary data to respond to the menu and item field data requests that follow the successful return of this function call. Inflexion UI may attempt to open several menus simultaneously. Therefore, further calls to IFXM - OpenMenu might occur before an open menu is closed with a call to IFXM - CloseMenu. Note however, that if the datasource URI used to open a menu is an exact match for one already open (for example, if the set of <link params> would be identical to an already-open menu) the runtime will NOT open a new menu instance (IFXM - OpenMenu will not be called). The current dynamic menu instance will instead be reused and shared between the menu components.



Caution

It is important that requests to IFXM - GetFieldData do not slow the Inflexion UI Runtime down. If there is any latency in the menu data's construction, it should be performed in IFXM - OpenMenu, rather than during calls to IFXM - GetFieldData.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_OpenMenu_<link prefix> (  
    IFX_HMODULE hModule,  
    IFX_HMENUID hMenuId,  
    IFX_HMENU* phMenu,  
    <link params>);
```

Parameters

- **hModule**
The module handle returned by IFXM - Initialize.
- **hMenuId**
The menu identifier used when the module returns a call to the Integration Layer regarding this menu.
- **phMenu**
A pointer to a variable into which the module places a menu handle. This value is returned to the module on every API call concerning the opened menu. The module implementer can use any value. The handle type is large enough to hold a pointer or an INT32. Typically, the handle is a pointer to a module's internal menu-specific data structure.
- **<link params>**
There may be additional parameters in this function, passing the link body, and the list of link parameters, as defined in the module definition file.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR**
Any error. The Inflexion UI Runtime does not perform any action, as if the clicked link is inactive. IFXM - CloseMenu is not called. You must free any memory allocated before returning this error.

Example

```
IFX_RETURN_STATUS IFXM_ifxuidemo_OpenMenu_menuPlugin(  
    IFX_HMODULE      hModule,  
    IFX_HMENUID      hMenuId,  
    IFX_HMENU        *phMenu,  
    const IFX_WCHAR  *body)  
{  
    IFX_INT32 i = 0;  
    CACHE_MENU* pLocalMenu;  
  
    // Allocate stack for task  
    if (NU_SUCCESS == NU_Allocate_Memory(&System_Memory,  
                                         (void **) &pLocalMenu,  
                                         sizeof(CACHE_MENU),  
                                         NU_NO_SUSPEND))  
    {  
        /* Populate menu cache & store hMenuId here */  
  
        *phMenu = (IFX_HMENU)pLocalMenu;  
  
        return IFX_SUCCESS;  
    }  
  
    return IFX_ERROR;  
}
```

Related Topics

[IFXM - CloseMenu](#)

[IFXM - GetFieldData](#)

IFXM - CloseMenu

This function is called when the specified menu is no longer required. It performs any required menu-specific cleanup.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_CloseMenu_<link prefix>
    (IFX_HMODULE hModule,
     IFX_HMENU hMenu);
```

Parameters

- **hModule**
The module handle returned by IFXM - Initialize.
- **hMenu**
The menu handle returned by IFXM - OpenMenu.

Return Values

- **IFX_SUCCESS**
Successful completion.

Example

```
IFX_RETURN_STATUS IFXM_ifxuidemo_CloseMenu_menuPlugin(
    IFX_HMODULE hModule,
    IFX_HMENU hMenu)
{
    CACHE_MENU* pLocalMenu = (CACHE_MENU*)hMenu;

    if (pLocalMenu)
    {
        /* free all allocated memory to data structure */
        NU_Deallocate_Memory(pLocalMenu);
    }

    return IFX_SUCCESS;
}
```

Related Topics

[IFXM - OpenMenu](#)

IFXM - GetItemCount

This function is called when a menu is successfully opened. It reports the total number of menu items.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_GetItemCount_<link prefix> (  
    IFX_HMODULE hModule,  
    IFX_HMENU hMenu,  
    IFX_INT32* pCount);
```

Parameters

- **hModule**
The module handle returned by IFXM - Initialize.
- **hMenu**
The menu handle returned by IFXM - OpenMenu.
- **pCount**
Points to an integer into which the module should put the count of items in this menu.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR**
Any error. The Inflexion UI Runtime treats the menu as having zero items.

Example

```
IFX_RETURN_STATUS IFXM_ifxuidemo_GetItemCount_menuPlugin(  
    IFX_HMODULE hModule,  
    IFX_HMENU hMenu,  
    IFX_INT32 *pCount)  
{  
    *pCount = ITEM_COUNT;  
    return IFX_SUCCESS;  
}
```

Related Topics

[IFXM - GetFirstActiveItem](#)

IFXM - GetMenuType

This function is called when the menu name is required. In the schema, the link type is associated with one menu node in the menu hierarchy, and may open that menu type, any descendent menu types, or any of the ancestor menu types. To simplify this, the Integrator application constructs an enum object containing all of the possible menu types that the link can open. This function must select one of these enum values and return it - the integration layer then returns the appropriate menu name to Inflexion UI Runtime. The following is the form of the enum:

```
enum IFXM_MENU_TYPE_<module name>_<link prefix>
{
    IFXM_MENU_TYPE_<module name>_<link prefix>_<menu name1>,
    IFXM_MENU_TYPE_<module name>_<link prefix>_<menu name2>,

    IFXM_MENU_TYPE_<module name>_<link prefix>_<menu nameN>
}
```

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_GetMenuType_<link prefix> (
    IFX_HMODULE hModule,
    IFX_HMENU hMenu,
    IFXM_MENU_TYPE_<module name>_<link prefix> *pType);
```

Parameters

- **hModule**
The module handle as initially returned by IFXM - Initialize.
- **hMenu**
The handle of the menu for which data is requested and returned by IFXM - OpenMenu.
- **pType**
Pointer to be filled in with one of the menu type enum values for this link type.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR**
Any error. The menu will not be opened.

Example

```
IFX_RETURN_STATUS IFXM_ifxuidemo_GetMenuType_menuPlugin(
    IFX_HMODULE hModule,
    IFX_HMENU hMenu,
    IFXM_MENU_TYPE_ifxuidemo_menuPlugin *pType)
```

```
{  
    *pType = IFXM_MENU_TYPE_ifxuidemo_menuPlugin_edit;  
    return IFX_SUCCESS;  
}
```

Related Topics

[IFXM - GetSortFieldSize](#)

[IFXM - GetSortFieldData](#)

[IFXM - GetSortDirection](#)

[IFXM - GetSortType](#)

[IFXM - GetPlaceholderSize](#)

[IFXM - GetPlaceholderData](#)

[IFXM - OpenMenu](#)

IFXM - GetLoadOnDemand

When a module menu contains a large number of items (for example, a contacts list with hundreds of contacts), opening that menu requires a significant amount of time (and resources) if Inflexion UI Runtime opened all the menu items at once. Inflexion UI Runtime supports a “load on demand” mode, that loads a subset of the menu items sufficient to fill the visible area of the menu (with a buffer region either side). Using “Load on Demand” prevents the menu being “sortable” (see the *Inflexion UI Express User’s Manual* section on “Load on Demand”).

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_GetLoadOnDemand_<link prefix> (  
    IFX_HMODULE      hModule,  
    IFX_HMENU        hMenu,  
    IFX_INT32*       pLoadOnDemand);
```

Parameters

- **hModule**
The module handle returned by IFXM - Initialize.
- **hMenu**
The menu handle for which data is requested and returned by IFXM - OpenMenu.
- **pLoadOnDemand**
Pointer to an IFX_INT32 into which 1 is placed if Load On Demand is used and 0 if not.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR**
Any error. In this case, Inflexion UI defaults to “Load On Demand will be supported”.

Example

```
IFX_RETURN_STATUS IFXM_ifxuidemo_GetLoadOnDemand_menuPlugin(  
    IFX_HMODULE      hModule,  
    IFX_HMENU        hMenu,  
    IFX_INT32        *pLoadOnDemand)  
{  
    *pLoadOnDemand = 0;  
    return IFX_SUCCESS;  
}
```

Related Topics

[IFXM - GetSortFieldSize](#)

[IFXM - GetSortDirection](#)

[IFXM - GetPlaceholderSize](#)

[IFXM - OpenMenu](#)

[IFXM - GetSortFieldData](#)

[IFXM - GetSortType](#)

[IFXM - GetPlaceholderData](#)

IFXM - GetPlaceholderSize

Deprecated - theme writers should use the `_placeholderActive` system variable.

IFXM - GetPlaceholderData

Deprecated - theme writers should use the `_placeholderActive` system variable.

IFXM - GetSortFieldSize

If the module would like the menu items to be sorted into some order other than menu item index order, it can specify one of the menu item fields to sort by, and whether to sort in ascending or descending order. If no sorting is required, the sort-related functions should return IFX_ERROR.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_GetSortFieldSize_<link prefix> (  
    IFX_HMODULE      hModule,  
    IFX_HREQUEST*    phRequest,  
    IFX_HMENU        hMenu,  
    IFX_INT32*       pSize);
```

Parameters

- **hModule**
The module handle returned by IFXM - Initialize.
- **phRequest**
Pointer to a variable into which the module may place a request handle. This value is returned to IFXM - GetSortFieldData. The module implementer can use any value. The handle type is large enough to hold a pointer or an IFX_INT32. The request handle stores the actual string to copy into the output buffer in IFXM - GetSortFieldData if the original string is likely to change.
- **hMenu**
The handle of the menu for which data is requested and returned by IFXM - OpenMenu.
- **pSize**
Pointer to an integer into which the module places the buffer size required to hold the field name, in characters, including the null terminator.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFXI_ERROR**
Any error. Sorting is not performed.

Example

```
IFX_RETURN_STATUS IFXM_ifxuidemo_GetSortFieldSize_menuPlugin(  
    IFX_HMODULE      hModule,  
    IFX_HREQUEST*    *phRequest,  
    IFX_HMENU        hMenu,  
    IFX_INT32        *pSize)
```

```
{  
    *pSize = (IFX_INT32)lc_wcslen(MENU_SORT_ALL);  
    return IFX_SUCCESS;  
}
```

Related Topics

[IFXM - GetSortFieldData](#)

[IFXM - GetSortDirection](#)

[IFXM - GetSortType](#)

[IFXM - GetLoadOnDemand](#)

[IFXM - OpenMenu](#)

IFXM - GetSortFieldData

If the module would like the menu items to be sorted into some order other than menu item index order, it can specify one of the menu item fields to sort by, and whether to sort in ascending or descending order. If no sorting is required, the sort-related functions should return `IFX_ERROR`.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_GetSortFieldData (
    IFX_HMODULE      hModule,
    IFX_HREQUEST     hRequest,
    IFX_WCHAR*       szData);
```

Parameters

- **hModule**
The module handle returned by `IFXM - Initialize`.
- **hRequest**
The request handle returned by `IFXM - GetSortFieldSize`.
- **szData**
Pointer to a buffer into which the module copies the field name. The buffer is large enough to hold the number of characters reported by the last call to `IFXM - GetSortFieldSize`. When `NULL`, the module cleans up any data allocated for `hRequest` and omits the string copy.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR**
Any error. Sorting is not be performed.

Example

```
IFX_RETURN_STATUS
IFXM_ifxuidemo_GetSortFieldData(
    IFX_HMODULE      hModule,
    IFX_HREQUEST     hRequest,
    IFX_WCHAR        *szData)
{
    if (szData)
        lc_wcscpy(szData, MENU_SORT_ALL);
    return IFX_SUCCESS;
}
```

Related Topics

[IFXM - GetSortFieldSize](#)

[IFXM - GetSortDirection](#)

[IFXM - GetSortType](#)

[IFXM - GetLoadOnDemand](#)

IFXM - GetSortDirection

This function is called when the menu sort direction is required. It specifies whether the sort is performed in ascending or descending order.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_GetSortDirection_<link prefix> (  
    IFX_HMODULE          hModule,  
    IFX_HMENU            hMenu,  
    IFX_SORT_DIRECTION*  pDirection);
```

Parameters

- **hModule**
The module handle returned by IFXM - Initialize.
- **hMenu**
The handle of the menu for which data is requested and returned by IFXM - OpenMenu.
- **pDirection**
Pointer to an enumeration into which IFX_SORT_ASCENDING or IFX_SORT_DESCENDING is placed.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR**
Any error. The sort is performed in ascending order.

Example

```
IFX_RETURN_STATUS IFXM_ifxuidemo_GetSortDirection_menuPlugin(  
    IFX_HMODULE          hModule,  
    IFX_HMENU            hMenu,  
    IFX_SORT_DIRECTION*  pDirection)  
{  
    *pDirection = IFX_SORT_ASCENDING;  
    return IFX_SUCCESS;  
}
```

Related Topics

[IFXM - GetSortFieldSize](#)[IFXM - GetFieldData](#)[IFXM - GetSortType](#)[IFXM - GetLoadOnDemand](#)[IFXM - OpenMenu](#)

IFXM - GetSortType

This function is called when the menu sort type is required. It specifies whether the sort is case sensitive.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_GetSortType_<link prefix> (  
    IFX_HMODULE      hModule,  
    IFX_HMENU        hMenu,  
    IFX_SORT_TYPE*   pType);
```

Parameters

- **hModule**
The module handle returned by IFXM - Initialize.
- **hMenu**
The handle of the menu for which data is requested and returned by IFXM - OpenMenu.
- **pType**
Pointer to an enumeration into which IFX_SORT_CASE_SENSITIVE or IFX_SORT_CASE_INSENSITIVE is placed.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR**
Any error. The sort is performed case sensitive.

Example

```
IFX_RETURN_STATUS IFXM_ifxuidemo_GetSortType_menuPlugin(  
    IFX_HMODULE      hModule,  
    IFX_HMENU        hMenu,  
    IFX_SORT_TYPE    *pType)  
{  
    *pType = IFX_SORT_CASE_SENSITIVE;  
    return IFX_SUCCESS;  
}
```

Related Topics

[IFXM - GetSortFieldSize](#)

[IFXM - GetSortFieldData](#)

[IFXM - GetSortDirection](#)

[IFXM - GetLoadOnDemand](#)

IFXM - GetFirstActiveItem

This function lets the module specify the first active item in the module menu. If pItem is filled with -1, or a number out of the range of the items in this menu, or if an error value is returned, the menu is filled according to the internal Inflexion UI Runtime logic.

Usage

```
IFX_RETURN_STATUS IFXM_<module>_GetFirstActiveItem_<menulink> (  
    IFX_HMODULE hModule,  
    IFX_HMENU hMenu,  
    IFX_INT32* pItem);
```

Parameters

- hModule
The module handle returned by IFXM - Initialize.
- hMenu
The handle of the menu for which data is requested and returned by IFXM - OpenMenu.
- pItem
Pointer to an integer to hold the (zero-based) index of the active item.

Return Values

- IFX_SUCCESS
Successful completion.
- IFX_ERROR
Any error. The menu is loaded according to the internal Inflexion UI Runtime logic.

Example

```
IFX_RETURN_STATUS IFXM_ifxuidemo_GetFirstActiveItem_menuPlugin(  
    IFX_HMODULE hModule,  
    IFX_HMENU hMenu,  
    IFX_INT32 *pItem)  
{  
    *pItem = 0;  
    return IFX_SUCCESS;  
}
```

Related Topics

[IFXM - SetActiveItem](#)

IFXM - SetActiveItem

This function notifies the module owning an open module menu when the active item changes.

Usage

```
IFX_RETURN_STATUS IFXM_<module>_SetActiveItem_<menulink> (  
    IFX_HMODULE hModule,  
    IFX_HMENU hMenu,  
    IFX_INT32 itemIndex);
```

Parameters

- **hModule**
The module handle returned by IFXM - Initialize.
- **hMenu**
The handle of the menu for which data is requested and returned by IFXM - OpenMenu.
- **itemIndex**
Pointer to an integer to hold the (zero-based) index of the active item.

Return Values

- **IFX_SUCCESS**
Always succeeds

Example

```
IFX_RETURN_STATUS IFXM_ifxuidemo_SetActiveItem_menuPlugin(  
    IFX_HMODULE hModule,  
    IFX_HMENU hMenu,  
    IFX_INT32 itemIndex)  
{  
    g_activeItem = itemIndex;  
    return IFX_SUCCESS;  
}
```

Related Topics

[IFXM - GetFirstActiveItem](#)

IFXI_RequestMenuRefresh

This function can be called at any time, either in the Inflexion UI execution context, or from a different context. It notifies the Inflexion UI Runtime that the contents of an open menu (whether visible) have changed. The callback returns immediately. The menu content is reread from the plugin and refreshed on screen immediately after the callback returns if the menu is on the active page or when the page containing the menu becomes active. This function should be used to notify the Inflexion UI Runtime when the number of items in a menu changes. If only the data associated with the items changes, it is more efficient to use IFXI_RequestFieldRefresh. This function can succeed if another module has exclusivity.

Usage

```
IFX_RETURN_STATUS IFXI_RequestMenuRefresh (  
    IFX_HMODULEID hModuleID,  
    IFX_HMENUID hMenuId);
```

Parameters

- **hModuleId**
The handle supplied to the module by IFXM - Initialize.
- **hMenuId**
The menu identifier to use when the module returns in the Integration Layer regarding the supplied menu context. If the field is not associated with a particular menu, pass NULL.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR**
The handle is invalid or the menu cannot be refreshed.
- **IFX_ERROR_ASYNC_BLOCKING**
The Inflexion UI Runtime is currently executing an asynchronous link and cannot process the request at this time
- **IFX_ERROR_INVALID_MODULEID**
The hModuleID parameter does not match an initialized module.
- **IFX_ERROR_ENGINE_NOT_READY**
The runtime is in the process of restarting following a critical error.

Related Topics

[IFXI_RequestFieldRefresh](#)

IFXI_RequestQueryActiveItem

This function can only be called from IFXM - ExecuteLink. It queries Inflexion UI about the active menu item in the menu identified by the parameter. This may be used by link functions attached to furniture buttons if they need to perform actions (such as play, delete, open) on the active item. A module can only query the active item for menus it manages. It cannot query the active item for static menus defined in XML. This function can succeed if another module has exclusivity.

Usage

```
IFX_RETURN_STATUS IFXI_RequestQueryActiveItem (  
    IFX_HMODULEID hModuleID,  
    IFX_HMENUID hMenuId,  
    IFX_INT32* pItem);
```

Parameters

- **hModuleId**
The handle supplied to the module by IFXM - Initialize.
- **hMenuId**
The menu identifier to use when the module returns to the Integration Layer regarding the supplied menu context.
- **pItem**
If an active item is found, it is filled with the index (0-based) of the menu item in the active slot.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR**
The menu handle is invalid or no menu item is active.
- **IFX_ERROR_ASYNC_BLOCKING**
The Inflexion UI engine is currently executing an asynchronous link and cannot process the request at this time.
- **IFX_ERROR_INVALID_MODULEID**
The hModuleID parameter does not match an initialized module.
- **IFX_ERROR_ENGINE_NOT_READY**
The runtime is in the process of restarting following a critical error.

-

Related Topics

[IFXI_RequestSetActiveItem](#)

[IFXM - ExecuteLink](#)

IFXI_RequestSetActiveItem

The module can ask that a specific item be made “active”, which will involve a sudden item reassignment in the Inflexion UI Runtime. This can be called in the Inflexion UI Runtime execution context or from a different context.

Usage

```
IFX_RETURN_STATUS IFXI_RequestSetActiveItem(  
    IFX_HMODULEID hModuleId,  
    IFX_HMENUID hMenuId,  
    IFX_INT32 item);
```

Parameters

- **hModuleId**
The handle supplied to the module by IFXM - Initialize.
- **hMenuId**
The menu identifier to use when the module returns to the Integration Layer regarding the supplied menu context.
- **item**
The (0-based) index of the item that should be made active where possible.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR**
The menu handle is invalid or no menu item is active.
- **IFX_ERROR_ASYNC_BLOCKING**
The Inflexion UI Runtime is currently executing an asynchronous link and cannot process the request at this time.
- **IFX_ERROR_INVALID_MODULEID**
The hModuleID parameter does not match an initialized module.
- **IFX_ERROR_ENGINE_NOT_READY**
The runtime is in the process of restarting following a critical error.

Related Topics

[IFXI_RequestQueryActiveItem](#)

Field Data

The following method templates are required to support field (for example, dynamic) tokens, which are used for nearly all typical module use cases.

- [IFXM - GetFieldStringSize](#)
- [IFXM - GetFieldData](#)
- [IFXM - SetFieldData](#)
- [IFXI_RequestFieldRefresh](#)

IFXM - GetFieldStringSize

The first function queries plug-in elements and event handlers created using the <link prefix> link string for string field data. The second function queries module menus created using the <link prefix> link name for string field data. The third function queries string field data defined globally within the module. If an hElement is available, the first function is always called first. If it did not expand the field, or no hElement was available, one of the final two functions is called depending on whether an hMenu is available and whether the field in question is defined globally or within a menu.



Note

This function is only used for string field data requests. It is not used when the field data defined as “int”, “float”, “boolean” or “time”.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_GetFieldStringSize_<link prefix> (  
    IFX_HMODULE hModule,  
    IFX_HREQUEST* phRequest,  
    IFX_HELEMENT hElement,  
    const IFX_WCHAR* szField,  
    IFX_INT32* pSize);
```

```
IFX_RETURN_STATUS IFXM_<module name>_GetFieldStringSize_<link prefix>_<field  
name> (  
    IFX_HMODULE hModule,  
    IFX_HREQUEST* phRequest,  
    IFX_HMENU hMenu,  
    IFX_INT32 item,  
    IFX_INT32* pSize);
```

```
IFX_RETURN_STATUS IFXM_<module name>_GetFieldStringSize_<field name> (  
    IFX_HMODULE hModule,  
    IFX_HREQUEST* phRequest,  
    IFX_INT32* pSize);
```

Parameters

- **hModule**
The module handle returned by IFXM - Initialize.
- **phRequest**
Pointer to a variable into which the module places a request handle. This value is returned to IFXM - GetFieldData. The module implementer can use any value. The handle type is large enough to hold a pointer or an IFX_INT32. The request handle stores the actual string to copy into the output buffer in IFXM - GetFieldData if the original string is likely to change.
- **hElement**
The handle of the element for which data is being requested and returned by IFXM - CreateElement if the field is associated with a plug-in element or event handler.
- **hMenu**
The handle of the menu for which data is requested and returned by IFXM - OpenMenu if the data is associated with a menu.
- **item**
The index (0-based) of the menu item for which data is requested if the data is associated with an item. Otherwise, it is -1.
- **szField**
The field name for which data is requested (for example, “artist” or “title”).
- **pSize**
Pointer to an integer into which the module puts the buffer size required to hold the field data, in characters, including the null terminator.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR**
Any error, including when the field name is not recognized by this module, or when a menu handle or item index are expected but not present. The field data is treated as having size zero.

Example

```
IFX_RETURN_STATUS IFXM_ifxuidemo_GetFieldStringSize_menuPlugin_icon(  
    IFX_HMODULE      hModule,  
    IFX_HREQUEST     *phRequest,  
    IFX_HMENU        hMenu,  
    IFX_INT32        item,  
    IFX_INT32        *pSize)  
{  
    CACHE_MENU* pMenu = (CACHE_MENU*) hMenu;
```

```
IFX_RETURN_STATUS retVal = IFX_ERROR;

if (pMenu && pSize && phRequest && ((IFX_UINT32)item < ITEM_COUNT) &&
(item >= 0))
{
    *pSize = (IFX_INT32)lc_wcslen(pMenu->menuData[item].iconAttr) + 1;
    *phRequest = (IFX_HREQUEST)pMenu->menuData[item].iconAttr;
    retVal = IFX_SUCCESS;
}

return retVal;
}
```

Related Topics

[IFXM - GetFieldData](#)

IFXM - GetFieldData

For string fields, this function retrieves field data immediately after each successful call to IFXM - GetFieldStringSize from which a non-zero size is returned. For the “int”, “float”, “boolean” or “time” fields, this function retrieves field data and is called at any time. In the first function, the module must return the exact string on which the size is based in the previous call to IFXM GetFieldStringSize. The request handle should store the actual string to copy into the output buffer if the original string is likely to change. Any memory allocated for the request handle must be cleaned up at this point.

The second function queries plug-in elements and event handlers created using the <link prefix> link string for non-string field data (something that only happens if the element definition in the module definition file has supportsFields=“true”). The third version queries module menus created using the <link prefix> link name for non-string field data. The last function queries string field data defined globally within the module. For non-string fields, if an hElement is available, the second function is always called first. If it did not expand the field, or no hElement was available, one of the final two functions is called depending on whether an hMenu is available and whether the field in question is defined globally or within a menu.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_GetFieldStringData (
    IFX_HMODULE hModule,
    IFX_HREQUEST hRequest,
    IFX_WCHAR* szData);

IFX_RETURN_STATUS IFXM_<module name>_GetField<field type>Data_<link prefix> (
    IFX_HMODULE hModule,
    IFX_HELEMENT hElement,
    const IFX_WCHAR* szField,
    <field type>* pData);

IFX_RETURN_STATUS IFXM_<module name>_GetField<field type>Data_<link
    prefix>_<field name> (
    IFX_HMODULE hModule,
    IFX_HMENU hMenu,
    IFX_INT32 item,
    <field type>* pData);

IFX_RETURN_STATUS IFXM_<module name>_GetField<field type>Data_<field name> (
```

```
IFX_HMODULE hModule,  
<field type>* pData);  
  
public abstract int getField<field type>Data_<field name>(  
    String szField,  
    Handle< <field type> > pData);  
  
public abstract int getField<field type>Data_<field name>(  
    int item,  
    Handle< <field type> > pData);  
  
public abstract int getField<field type>Data_<field name>(  
    Handle< <field type> > pData);
```

Parameters

- **hModule**
The module handle returned by IFXM - Initialize.
- **hRequest**
The request handle returned by IFXM - GetFieldStringSize.
- **hElement**
The handle of the element for which data is requested and returned by IFXM - CreateElement if the field is associated with a plug-in element or event handler.
- **hMenu**
The handle of the menu for which data is requested and returned by IFXM - OpenMenu if the data is associated with a menu.
- **item**
The index (0-based) of the menu item for which data is requested if the data is associated with an item. Otherwise, -1.
- **szField**
The field name for which data is requested (for example “artist” or “title”).
- **szData**
Pointer to a buffer into which the module copies the field data. The buffer is large enough to hold the number of characters reported by the last call to IFXM - GetFieldStringSize. When NULL, the module cleans up any data allocated for hRequest and omits the string copy.

- **pData**
Pointer to a buffer into which the module copies the field data. The <field type> of pData is IFX_INT32*, float, IFX_BOOL*, or IFX_TIME*.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR**
Any error, including when the field name is not recognized by this module, or when a menu handle or item index are expected but not present.

Example

```
IFX_RETURN_STATUS IFXM_ifxuidemo_GetFieldStringData(  
    IFX_HMODULE      hModule,  
    IFX_HREQUEST     hRequest,  
    IFX_WCHAR        *szData)  
{  
    if (szData)  
        lc_wcscpy(szData, (IFX_WCHAR*)hRequest);  
    return IFX_SUCCESS;  
}
```

Related Topics

[IFXM - GetFieldStringSize](#)

IFXM - SetFieldData

These functions are used to notify modules when “input” fields change value within Inflexion UI Runtime.

The first function notifies the module owning the plug-in element or event handler created using the <link prefix> link string (something that will only happen if the element definition in the module definition file has supportsFields=“true”). The second version notifies the module owning the module menu created using the <link prefix> link name item or menu context data. The last function notifies the module owning the global field data. If an hElement handle is available, the first function is always called first. If the module owning the element does not return IFX_SUCCESS, or no hElement was available, one of the final two functions is called depending on whether an hMenu is available and whether the field in question is defined globally or within a menu.

Note that only “int”, “boolean”, and “float” fields can be input fields, so these functions are never generated for “time” or “string” fields.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_SetField<field type>Data_<link prefix> (  
    IFX_HMODULE hModule,  
    IFX_HELEMENT hElement,  
    const IFX_WCHAR* szField,  
    <field type> value,  
    IFX_INT32 final);
```

```
IFX_RETURN_STATUS IFXM_<module name>_SetField<field type>Data <link prefix>  
    <field name> (  
    IFX_HMODULE hModule,  
    IFX_HMENU hMenu,  
    IFX_INT32 item,  
    <field type> value,  
    IFX_INT32 final);
```

```
IFX_RETURN_STATUS IFXM_<module name>_SetField<field type>Data_<field name> (  
    IFX_HMODULE hModule,  
    <field type> value,  
    IFX_INT32 final);
```

Parameters

- **hModule**
The module handle returned by IFXM - Initialize.
- **hElement**
The handle of the element for which data is requested and returned by IFXM - CreateElement if the field is associated with a plug-in element or event handler.
- **hMenu**
The handle of the menu for which data is requested and returned by IFXM - OpenMenu if the data is associated with a menu.
- **item**
The index (0-based) of the menu item for which data is requested if the data is associated with an item. Otherwise, -1.
- **szField**
The field name for which data is requested (for example “artist” or “title”).
- **value**
New value of the field.
- **final**
If set to 1, Inflexion UI Runtime does not expect this field to change value in the immediate future. If set to 0, the field is expected to change value again shortly (for example, if the field is attached to a drag region element that has been flicked, or is processing a drag event).

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR**
Any error, including when the field name is not recognized by this module, or when a menu handle or item index is expected, but not present.

Example

```
IFX_RETURN_STATUS IFXM_ifxuidemo_SetFieldIntData_globalField1(  
    IFX_HMODULE hModule,  
    IFX_INT32 value,  
    IFX_INT32 final)  
{  
    /* Apply new value */  
    onGlobalField1Change(value);  
  
    if (final)  
    {  
        /* Cache value for future use */  
    }  
}
```

```
    g_CachedGlobalField = value;
}

return IFX_SUCCESS;
}
```

Related Topics

[IFXM - GetFieldStringSize](#)

[IFXM - GetFieldData](#)

IFXI_RequestFieldRefresh

This function can be called at any time, either in the Inflexion UI execution context or from a different context. It notifies the Inflexion UI Runtime that field data is changed and the display must be refreshed. The function returns immediately. The screen is updated as soon as possible, rereading the specified field data from the module in the process. This function can succeed if another module has exclusivity.

Usage

```
IFX_RETURN_STATUS IFXI_RequestFieldRefresh (  
    IFX_HMODULEID hModuleID,  
    IFX_HMENUID hMenuId,  
    IFX_INT32 item,  
    const IFX_WCHAR* szField);
```

Parameters

- **hModuleId**
The handle supplied to the module by IFXM - Initialize.
- **hMenuId**
The menu identifier to use when the module returns to the Integration Layer regarding the supplied menu context. If the field is not associated with a particular menu, pass NULL.
- **item**
The index (0-based) of the menu item for which the field data is changed. If the data is not associated with a specific item, pass -1.
- **szField**
The name of the field whose associated data is changed. An empty string ("") may be passed here to indicate that all fields associated with the specified item or menu have changed. The field name buffer cannot exceed IFX_MAX_FIELD_NAME_LENGTH in length, inclusive of NULL terminators.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR_ASYNC_BLOCKING**
The Inflexion UI Runtime is currently executing an asynchronous link and cannot process the request at this time.
- **IFX_ERROR_INVALID_MODULEID**
The hModuleID parameter does not match an initialized module.
- **IFX_ERROR_ENGINE_NOT_READY**
The runtime is in the process of restarting following a critical error.

Example

```
IFXI_RequestFieldRefresh (g_moduleID, NULL, -1, L"result");
```

Related Topics

[IFXI_RequestMenuRefresh](#)

Plugin

The following method templates are related to supporting plugin elements of type “event handler” or “graphic”:

- [IFXM - CreateElement](#)
- [IFXM - DestroyElement](#)
- [IFXM - ActivateElement](#)
- [IFXM - DeactivateElement](#)
- [IFXM - SetElementFocus](#)
- [IFXM - UnsetElementFocus](#)
- [IFXM - ProcessElementKeyDownEvent](#)
- [IFXM - ProcessElementKeyUpEvent](#)
- [IFXM - GetElementCaretPosition](#)
- [IFXI_RequestRefreshBufferedElement](#)
- [IFXI_RequestResizeBufferedElementBuffer](#)
- [IFXI_RequestFullScreenStart](#)
- [IFXI_RequestFullScreenStop](#)

The following method templates are only ever implemented for plugin elements of type “graphic” type:

- [IFXM - ChangeElementMode](#)
- [IFXM - ProcessElementStylusDragEvent](#)
- [IFXM - ProcessElementStylusCancelEvent](#)
- [IFXM - ProcessElementStylusUpEvent](#)
- [IFXM - ProcessElementStylusDownEvent](#)
- [IFXM - PositionElement](#)
- [IFXM - PaintElement](#)

**Note**

The precise set of methods required to be implemented for an element from those listed above depends on the options selected in the module definition file entry for the plugin element.

IFXM - CreateElement

This function creates an instance of an element defined in the module definition file. The second version of this function is used for event handlers, because they do not need pProperty. Note that if the eventhandler URI used to create an element is an exact match for one already open (for example, if the set of <link params> would be identical to an already-created element) the runtime will NOT create a new element instance (IFXM - CreateElement will not be called). The current element instance will instead be reused and shared between the plugin elements.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_CreateElement_<link prefix> (  
    IFX_HMODULE          hModule,  
    IFX_HELEMENTID       hElementId,  
    IFX_HELEMENT*        phElement,  
    IFX_HMENUID          hMenuId,  
    IFX_HMENU            hMenu,  
    IFX_INT32            item,  
    IFX_ELEMENT_PROPERTY* pProperty,  
    <link params>);
```

```
IFX_RETURN_STATUS IFXM_<module name>_CreateElement_<link prefix> (  
    IFX_HMODULE          hModule,  
    IFX_HELEMENT*        phElement,  
    IFX_HMENUID          hMenuId,  
    IFX_HMENU            hMenu,  
    IFX_INT32            item,  
    <link params>);
```

Parameters

- hModule
The module handle returned by IFXM - Initialize.
- hElementId

The element identifier used when the module returns to the Integration Layer regarding this element. In particular note that the full screen start/stop and buffer refresh request IFXI_* functions will require this value and NOT the element handle returned in phElement. This is only supplied for graphic elements.

- **phElement**

Pointer to a variable into which the module places an element handle. This value is returned to the module on every API call concerning the element. The module implementer can use any value. The handle type is guaranteed large enough to hold a pointer or an IFX_INT32. Typically, the handle is a pointer to an element-specific data structure internal to the module.

- **hMenuId**

The menu identifier used when the module returns to the Integration Layer regarding the supplied menu context.

- **hMenu**

The handle of the menu for which data is requested and returned by IFXM - OpenMenu if the data is associated with a menu. Otherwise, it is NULL.

- **item**

The index (0-based) of the menu item for which data is requested if the data is associated with an item. Otherwise, it is -1.

- **pProperty**

Out parameter pointing to an IFX_ELEMENT_PROPERTY structure to be filled in by the module indicating which modes the module supports for this type of plugin element and other configuration details, such as whether the plugin element buffer contains translucent pixels or not.

The IFX_MODE_ELEMENT_BUFFERED_AND_DIRECT, IFX_MODE_ELEMENT_OPENGL_TEXTURE, and IFX_MODE_ELEMENT_BUFFERED modes are supported (if the module supports more than one mode and permits the modes to be dynamically switched, the values should be ORed together). If the module supports buffered mode, it can inspect the requiredBufferWidth, requiredBufferHeight, and requiredBufferFormat members of this structure. The requiredFormat member can only be set when in one of the OpenGL rendering modes. These are set to the forced extent of the plugin element (if the element has an extentHint property set) or -1 if there is no forced extent. The module can set the requiredBufferWidth and requiredBufferHeight members to change the initial buffer size of the element. Note that Inflexion UI will scale the buffer into any extent set by the theme designer automatically (or into an extent hint rectangle whilst preserving aspect ratio of the buffer, if extent hint is used and no extent is set). If there is no extent or extent hint used in the current layout, Inflexion UI will render the buffer to screen unscaled. If the element contains any transparent or partially-transparent pixels, it should set the “translucency” parameter to “1”, otherwise the runtime will assume the element is opaque. If the render mode is “Direct OpenGL 2.0” and the element has transparent or partially-transparent pixels where the transparency is already factored into the color channels (“Pre-multiplied alpha”), the “hasPreMultipliedAlpha” member should be set to “1”.

- <link params>
A sequence of parameters matching the link body and any link parameters defined in the module definition file for this element.

Return Values

- IFX_SUCCESS
Successful completion.
- IFX_ERROR
Any error. The element does not appear on screen. IFXM - DestroyElement is not called, therefore all resources must be freed before returning.

Example

```
IFX_RETURN_STATUS IFXM_ifxuimmcore_CreateElement_video(  
    IFX_HMODULE          hModule,  
    IFX_HELEMENTID       hElementId,  
    IFX_HELEMENT         *phElement,  
    IFX_HMENUID          hMenuId,  
    IFX_HMENU            hMenu,  
    IFX_INT32            item,  
    IFX_ELEMENT_PROPERTY *pProperty,  
    const IFX_WCHAR      *body,  
    const IFX_WCHAR      *paramSource,  
    const IFX_WCHAR      *paramLoop,  
    const IFX_WCHAR      *paramFullScreen,  
    const IFX_WCHAR      *paramStretchMode)  
{  
    IFX_RETURN_STATUS      retVal = IFX_ERROR;  
    IFX_MMC_VIDEO_PLAYER_DATA* pNewObject = 0;  
  
    IFXI_MMC_UNUSED_PARAMETER(hModule);  
    IFXI_MMC_UNUSED_PARAMETER(hMenuId);  
    IFXI_MMC_UNUSED_PARAMETER(hMenu);  
    IFXI_MMC_UNUSED_PARAMETER(item);  
  
    if (body != NU_NULL && wcslen(body) > 0)  
    {  
        /* deal with body */  
    }  
  
    if(paramSource && wcslen(paramSource) > 0)  
    {  
        /* deal with parameter */  
    }  
  
    if(paramLoop && wcslen(paramLoop) > 0)  
    {  
        /* deal with parameter */  
    }  
  
    if(paramFullScreen && wcslen(paramFullScreen) > 0)  
    {  
        /* deal with parameter */  
    }  
}
```

```
    }

    if(paramStretchMode && wcslen(paramStretchMode) > 0)
    {
        /* deal with parameter */
    }

    /* create element instance*/

    pProperty->supportedModes = IFX_MODE_ELEMENT_BUFFERED_AND_DIRECT;
    *phElement = (IFX_HELEMENT)pNewObject;

    retVal = IFX_SUCCESS;

    return (retVal);
```

Related Topics

[IFXM - ActivateElement](#)

[IFXM - DestroyElement](#)

IFXM - DestroyElement

This function is called when the specified plugin element or event handler is no longer required because the page on which it is displayed is being closed.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_DestroyElement_<link prefix> (  
    IFX_HMODULE      hModule,  
    IFX_HELEMENT     hElement);
```

Parameters

- **hModule**
The module handle returned by IFXM - Initialize.
- **hElement**
The handle of the element for which data is requested and returned by IFXM_CreateElement if the field is associated with a plugin element or event handler.

Return Values

- **IFX_SUCCESS**
Successful completion.

Description

Once an element is destroyed using this function, the element cannot paint to the screen or generate callbacks to Inflexion UI, even if some part of the destruction process fails. There is no failure code to return. The Inflexion UI Runtime always behaves as if destruction was successful.

Example

```
IFX_RETURN_STATUS IFXM_ifxuimmcore_DestroyElement_video(  
    IFX_HMODULE      hModule,  
    IFX_HELEMENT     hElement)  
{  
    IFX_MMC_VIDEO_PLAYER_DATA* pObject =  
        (IFX_MMC_VIDEO_PLAYER_DATA*) hElement;  
  
    IFXI_MMC_UNUSED_PARAMETER(hModule);  
  
    if (pObject != NU_NULL)  
    {  
        /* destroy element and release resources */  
    }  
  
    return (IFX_SUCCESS);  
}
```

Related Topics

[IFXM - DeactivateElement](#)

[IFXM - CreateElement](#)

IFXM - ChangeElementMode

This function switches a plugin element to a new rendering mode and passes the appropriate rendering context to the module.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_ChangeElementMode_<link prefix> (  
    IFX_HMODULE          hModule,  
    IFX_HELEMENT         hElement,  
    IFX_ELEMENT_MODE     newMode,  
    void*                 pContext),
```

Parameters

- **hModule**
The module handle returned by IFXM - Initialize.
- **hElement**
The handle of the element for which data is requested and returned by IFXM - CreateElement if the field is associated with a plugin element or event handler.
- **newMode**
Required new mode for element. This is one of the modes the module registered as supported in the IFX_ELEMENT_PROPERTY structure in IFXM - CreateElement.
- **pContext**
Pointer to a context object containing information required by the module to render the element for the current element mode. The precise structure of this context varies with mode and platform.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR**
Any error. The element node is unchanged.

Description

If the new mode is IFX_MODE_ELEMENT_BUFFERED_AND_DIRECT and the element is activated, the module renders to the appropriate client area immediately following the next call to IFXM - PositionElement. The module stores the pointer to the context structure. It does not

cache the value of the device-specific context within the structure, as this may change during the lifetime of the element in the current mode.

- **IFX_MODE_ELEMENT_BUFFERED_AND_DIRECT**

For this mode, `pContext` points to a `IFX_DIRECT_RENDER_CONTEXT` structure, with `pDisplay` pointing to an `IFX_DISPLAY` structure (defined in `nucleus/os/include/ui/ifxui_defs.h`). In most cases, the direct mode element is interested in the internal member of the `IFX_DISPLAY` structure, which is a generic pointer to some sort of platform-specific rendering context that the module can use to create its own rendering context with which to draw to screen. Because it is likely the rendering context passed into this method belongs to the Inflexion UI Runtime, modifying it is not advised.

See the implementation of the porting layer for the chosen platform for further details on what the internal member points to, and how it should be used.

- **IFX_MODE_ELEMENT_OPENGL_TEXTURE**

For this mode, `pContext` points to a `IFX_OPENGL_RENDER_CONTEXT` structure. The width and height members are populated by the engine to provide the native dimensions (if applicable). The texture member must be populated with a valid OpenGL texture ID. The `tex_width` and `tex_height` members must represent the texture size and the width and height members must represent the bitmap size within that texture.

The EGL context information is also provided for cases where this is required (creating an `EGLImage` for example).

The member `textureTarget` indicates the target to which the texture will be bound just before it is drawn, and should be one member of the `GLenum` set valid to be passed to `glBindTexture`. If not changed by the module during the call to `changeElementMode`, the target used will be `GL_TEXTURE_2D`.

The plugin element may optionally provide up to three additional OpenGL texture ids in the `secondaryTextures` member. If the render mode is Direct OpenGL 2.0, these will be available to any custom shader using the Inflexion intrinsics

`IFX_SECONDARY_TEXTURE_0`, `IFX_SECONDARY_TEXTURE_1` or `IFX_SECONDARY_TEXTURE_2`. Typically, this will be used by, for example, video plugin elements using YUV decoded data, with the custom shader converting the YUV data to RGB data.

- **IFX_MODE_ELEMENT_BUFFERED**

For this mode, `pContext` points to a `IFX_BUFFERED_RENDER_CONTEXT` structure, this structure must have the rendering buffer pointer populated by the plugin no later than the first time `PaintElement` returns. This pointer can be updated by the module during any subsequent paint element calls. The format of the buffer is indicated by the engine in the `format` member (defaults to `RGBA`), along with the height and width of the buffer. On each call to `IFXM - PaintElement`, the module must render an image into this buffer (the location and dimensions of which may change between calls to `IFXM - PaintElement`).

This method is not generated for event handler type elements.

Example

```
IFX_RETURN_STATUS IFXM_ifxuimmcore_ChangeElementMode_video(  
    IFX_HMODULE      hModule,  
    IFX_HELEMENT     hElement,  
    IFX_ELEMENT_MODE newMode,  
    VOID             *pContext)  
{  
    IFX_RETURN_STATUS retVal = IFX_ERROR;  
    IFX_MMC_VIDEO_PLAYER_DATA *pNewObject =  
                                        (IFX_MMC_VIDEO_PLAYER_DATA*)  
hElement;  
  
    IFXI_MMC_UNUSED_PARAMETER(hModule);  
    IFXI_MMC_UNUSED_PARAMETER(newMode);  
  
    /* check new mode */  
  
    /* take copy of context */  
  
    /* create element rendering context from Inflexion UI context in  
       direct mode */  
  
    return (IFX_SUCCESS);  
}
```

Related Topics

[IFXM - CreateElement](#)

[IFXM - ChangeElementMode](#)

[IFXM - PositionElement](#)

[IFXM - PaintElement](#)

IFXM - ActivateElement

Inflexion UI calls this function to indicate that the specified element should be active and painted to the screen as necessary depending on its type. This function reactivates an element previously deactivated by calling IFXM - DeactivateElement.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_ActivateElement_<link prefix> (  
    IFX_HMODULE      hModule,  
    IFX_HELEMENT     hElement);
```

Parameters

- **hModule**
The module handle returned by IFXM - Initialize.
- **hElement**
The handle of the element for which data is requested and returned by IFXM - CreateElement if the field is associated with a plugin element or event handler.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR**
Any error.

Description

Inflexion UI calls this function to indicate that the specified element should be active and painted to the screen as necessary depending on its type. This indicates that elements of type IFX_MODE_ELEMENT_BUFFERED_AND_DIRECT should start to paint to the screen using the required hardware resources as necessary. Elements of type IFX_MODE_ELEMENT_BUFFERED and IFX_MODE_ELEMENT_OPENGL_TEXTURE draw to the screen during the Inflexion UI Runtime rendering cycle. All buffered mode elements must call IFXI_RequestRefreshBufferedElement each time the module determines the buffer contents have changed.

Example

```
IFX_RETURN_STATUS IFXM_ifxuimmcore_ActivateElement_video(  
    IFX_HMODULE      hModule,  
    IFX_HELEMENT     hElement)  
{  
    IFX_RETURN_STATUS      retVal = IFX_ERROR;  
    IFX_MMC_VIDEO_PLAYER_DATA* pObject =  
        (IFX_MMC_VIDEO_PLAYER_DATA*) hElement;  
  
    IFXI_MMC_UNUSED_PARAMETER(hModule);
```

```
        /* direct mode elements are now free to draw to the screen */  
        return (retVal);  
    }
```

Related Topics

[IFXM - DeactivateElement](#)

[IFXI_RequestRefreshBufferedElement](#)

IFXM - DeactivateElement

Inflexion UI Runtime calls this function when the specified element should no longer be painted to the screen. This can be a temporary state, and the element can be reactivated later.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_DeactivateElement_<link prefix> (  
    IFX_HMODULE      hModule,  
    IFX_HELEMENT     hElement);
```

Parameters

- **hModule**
The module handle returned by IFXM - Initialize.
- **hElement**
The handle of the element for which data is requested and returned by IFXM - CreateElement if the field is associated with a plugin element or event handler.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR**
Any error. If the element cannot be deactivated or does not stop rendering, Inflexion UI may cease to function correctly.

Description

For buffered mode elements, no further IFXM - PaintElement calls are issued. All calls to IFXI_RequestRefreshBufferedElement by this element are ignored.

No frames should be drawn after this function returns for elements of type IFX_MODE_ELEMENT_BUFFERED_AND_DIRECT.

The element implementation can continue to stream data from a data source. It cannot attempt to render data until Inflexion UI reactivates the object with a call to IFXM - ActivateElement.

Example

```
IFX_RETURN_STATUS IFXM_ifxuimcore_DeactivateElement_video(  
    IFX_HMODULE      hModule,  
    IFX_HELEMENT     hElement)  
{  
    IFX_RETURN_STATUS      retVal = IFX_ERROR;  
    IFX_MMC_VIDEO_PLAYER_DATA* pObject =  
        (IFX_MMC_VIDEO_PLAYER_DATA*) hElement;  
  
    IFXI_MMC_UNUSED_PARAMETER(hModule);
```

```
        /* direct mode elements should stop drawing to screen */  
        return (retVal);  
    }
```

Related Topics

[IFXM - ActivateElement](#)

[IFXI_RequestRefreshBufferedElement](#)

[IFXM - PaintElement](#)

IFXM - SetElementFocus

Inflexion UI calls this function when an element is given the input focus. Key events will be forwarded to the module, and only handled by Inflexion UI Runtime if the module declines to consume them. Touch events can be routed to a stylus supporting module element even if focus has not been set.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_SetElementFocus_<link prefix> (  
    IFX_HMODULE      hModule,  
    IFX_HELEMENT     hElement);
```

Parameters

- **hModule**
The module handle returned by IFXM - Initialize.
- **hElement**
The element's handle for which data is being requested. It is returned by IFXM - CreateElement when the field is associated with a plugin element or event handler.

Return Values

- **IFX_SUCCESS**
Successful completion.

Description

The function is only called when the element in question is a graphical plugin or an Inflexion UI Runtime element with an associated event handler. A call to IFXM - UnsetElementFocus removes the element's focus from this element.

Example

```
IFX_RETURN_STATUS IFXM_ifxuimmcore_SetElementFocus_video(  
    IFX_HMODULE      hModule,  
    IFX_HELEMENT     hElement)  
{  
    IFX_MMC_VIDEO_PLAYER_DATA*  pObject =  
                                (IFX_MMC_VIDEO_PLAYER_DATA*) hElement;  
  
    IFXI_MMC_UNUSED_PARAMETER(hModule);  
  
    /* note the element has the input focus */  
  
    /* refresh any global fields that refer to the element in focus */  
  
    return (IFX_SUCCESS);  
}
```

Related Topics

[IFXM - UnsetElementFocus](#)

IFXM - UnsetElementFocus

Inflexion UI calls this function when an element has the user input focus removed by Inflexion UI Runtime. It is only called when the element in question is a graphical plugin or an Inflexion UI Runtime element with an associated event handler.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_UnsetElementFocus_<link prefix> (  
    IFX_HMODULE      hModule,  
    IFX_HELEMENT     hElement);
```

Parameters

- **hModule**
The module handle returned by IFXM - Initialize.
- **hElement**
The element's handle for which data is being requested. It is returned by IFXM - CreateElement when the field is associated with a plugin element or event handler.

Return Values

- **IFX_SUCCESS**
Successful completion.

Example

```
IFX_RETURN_STATUS IFXM_ifxuimmcore_UnsetElementFocus_video(  
    IFX_HMODULE      hModule,  
    IFX_HELEMENT     hElement)  
{  
    IFX_MMC_VIDEO_PLAYER_DATA*  pObject =  
                                (IFX_MMC_VIDEO_PLAYER_DATA*) hElement;  
  
    IFXI_MMC_UNUSED_PARAMETER(hModule);  
  
    /* note the element has lost the input focus */  
  
    /* refresh any global fields that refer to the element that was in  
    focus */  
  
    return (IFX_SUCCESS);  
}
```

Related Topics

[IFXM - SetElementFocus](#)

IFXM - ProcessElementKeyDownEvent

Inflexion UI calls this function when a key down event is received and there is a focused element. It is only called when the element in question is a graphical plugin or an Inflexion UI Runtime element with an associated event handler.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_ProcessElementKeyDownEvent_<link  
    prefix> (  
    IFX_HMODULE      hModule,  
    IFX_HELEMENT     hElement,  
    IFX_INT32        key,  
    IFX_INT32*       pConsumed) ;
```

Parameters

- **hModule**
The module handle returned by IFXM - Initialize.
- **hElement**
The element's handle for which data is being requested. It is returned by IFXM - CreateElement when the field is associated with a plugin element or event handler.
- **key**
Key scan code or one of the Inflexion UI pre-defined key values.
- **pConsumed**
Returns an IFX_INT32 indicating whether the key was consumed. 1 means it was consumed, 0 means it was not consumed.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR**
Any error. The Inflexion UI Runtime processes the event as though it was not consumed.

Description

A call to IFXM - SetElementFocus sets the element's focus prior to calling this function. The Inflexion UI Runtime only sends key events to an element that is the focus.

This function returns a Boolean value, pConsumed, indicating whether the key is consumed. When false (0) is returned, the Inflexion UI Runtime processes the event using the triggers set up in the page template. When true (1) is returned, the Inflexion UI Runtime does nothing more.

The plugin can also return false (0), even if the key event is handled, allowing additional event processing by the Inflexion UI Runtime.

Inflexion UI does not use key up events internally. Therefore, the element must declare whether it consumes the event during this function call, even though it may not do anything until the key up event occurs. IFXM - ProcessElementKeyUpEvent is called after IFXM - ProcessElementKeyDownEvent, unless the element loses its focus or is destroyed. It is possible for more than one different key down event to be received before the corresponding key up event.

Pre-defined Inflexion UI Runtime key codes are defined as:

```
enum tagIFX_KEY_CODE
{
    IFX_KEY_CODE_HASH = 0x0000FFFF,
    IFX_KEY_CODE_ASTERISK,
    IFX_KEY_CODE_UP,
    IFX_KEY_CODE_DOWN,
    IFX_KEY_CODE_LEFT,
    IFX_KEY_CODE_RIGHT,
    IFX_KEY_CODE_LSOFT,
    IFX_KEY_CODE_RSOFT,
    IFX_KEY_CODE_SELECT,
    IFX_KEY_CODE_BACK
};
typedef enum tagIFX_KEY_CODE IFX_KEY_CODE;
```

All other keys are passed into this function with their original scan code as the “key” parameter. Inflexion UI assumes that the scan codes for a-z, A-Z and 0-9 use their ASCII value as the scan code.

Example

```
IFX_RETURN_STATUS IFXM_ifxuimmcore_ProcessElementKeyDownEvent_video(
    IFX_HMODULE      hModule,
    IFX_HELEMENT     hElement,
    IFX_INT32        key,
    IFX_INT32        *pConsumed)
{
    IFX_MMC_VIDEO_PLAYER_DATA* pObject =
        (IFX_MMC_VIDEO_PLAYER_DATA*) hElement;

    IFXI_MMC_UNUSED_PARAMETER(hModule);

    /* handle key */

    /* tell the Inflexion UI Runtime that the event has been consumed */
    *pConsumed = 1;

    return (IFX_SUCCESS);
}
```

Related Topics

[IFXM - SetElementFocus](#)

[IFXM - ProcessElementKeyUpEvent](#)

IFXM - ProcessElementKeyUpEvent

Inflexion UI calls this function when a key-up event is received and there is a focused element. It is only called when the element in question is a graphical plugin, or an Inflexion UI element with an associated event handler.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_ProcessElementKeyUpEvent  
  
<link prefix> (  
    IFX_HMODULE      hModule,  
    IFX_HELEMENT     hElement,  
    IFX_INT32        key);
```

Parameters

- **hModule**
The module handle returned by IFXM - Initialize.
- **hElement**
The element's handle for which data is being requested. It is returned by IFXM - CreateElement when the field is associated with a plug-in element or event handler.
- **key**
Key scan code or one of the Inflexion UI Runtime's pre-defined key values.

Return Values

- **IFX_SUCCESS**
Successful completion.

Description

Inflexion UI Runtime calls this function when a key-up event is received and there is a focused element. It is only called when the element in question is a graphical plug-in, or an Inflexion UI element with an associated event handler. A call to IFXM - SetElementFocus sets the element's focus prior to calling this function. The Inflexion UI Runtime only sends key events to an element that is the focus.

Inflexion UI Runtime does not use key up events internally. Therefore, the element declares whether it consumes the event during the key down event, even though it may not do anything until the key up event occurs. IFXM - ProcessElementKeyUpEvent is called after IFXM - ProcessElementKeyDownEvent, unless the element loses its focus or is destroyed. It is possible

for more than one different key down event to be received before the corresponding key up event.

The topic “[IFXM - ProcessElementKeyDownEvent](#)” describes key codes in further detail.

All other keys are received as their original scan code. The Inflexion UI Runtime assumes that the scan codes for a-z, A-Z and 0-9 use their ASCII value as the scan code.

Example

```
IFX_RETURN_STATUS IFXM_ifxuimmcore_ProcessElementKeyUpEvent_video(  
    IFX_HMODULE      hModule,  
    IFX_HELEMENT     hElement,  
    IFX_INT32        key)  
{  
    IFX_MMC_VIDEO_PLAYER_DATA* pObject =  
        (IFX_MMC_VIDEO_PLAYER_DATA*) hElement;  
  
    IFXI_MMC_UNUSED_PARAMETER(hModule);  
  
    /* handle key up */  
  
    return (IFX_SUCCESS);  
}
```

Related Topics

[IFXM - ProcessElementKeyDownEvent](#)

IFXM - ProcessElementStylusDragEvent

This function is called when a stylus drag event is detected following a prior successful call to ProcessElementDownEvent.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_ProcessElementStylusDragEvent_<link  
prefix> (  
    IFX_HMODULE hModule,  
    IFX_HELEMENT hElement,  
    IFX_INT32 x,  
    IFX_INT32 y);
```

Parameters

- **hModule**
The module handle returned by IFXM - Initialize.
- **hElement**
The element's handle for which data is being requested. It is returned by IFXM_CreateElement when the field is associated with a plug-in element or event handler.
- **x**
The x coordinate of the stylus event.
- **y**
The y coordinate of the stylus event.

Return Values

- **IFX_SUCCESS**
The element wants exclusive access to the drag event.
- **IFX_ERROR**
The element does not want exclusive access to the drag event.

Description

If the module did not consume the event on the ProcessElementDownEvent, it should decide whether it wants exclusive access to the stylus drag, and return that decision to the Inflexion UI Runtime through the return value. If IFX_SUCCESS is returned, no other Inflexion UI elements will receive drag events until the current drag event ends. If IFX_ERROR is returned, other Inflexion UI elements will continue to receive drag events (and may subsequently grab the drag focus).

The given coordinate is a pixel position from the top-left corner of the plug-in element. 0,0 is assumed to be the top-left corner, with positive x and y axis extending right and down, respectively.

Related Topics

[IFXM - ProcessElementStylusCancelEvent](#)

[IFXM - ProcessElementStylusDownEvent](#)

[IFXM - ProcessElementStylusUpEvent](#)

IFXM - ProcessElementStylusCancelEvent

This function is called when a stylus down or drag state is terminated. This may happen if some other Inflexion UI Runtime element acquires the exclusive drag focus (for example, if a relative drag region exceeds its drag threshold) or some event occurs to negate the “mouse down” Inflexion UI Runtime state.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_ProcessElementStylusCancelEvent  
<link prefix> (  
    IFX_HMODULE hModule,  
    IFX_HELEMENT hElement);
```

Parameters

- **hModule**
The module handle returned by IFXM - Initialize.
- **hElement**
The element’s handle for which data is being requested. It is returned by IFXM - CreateElement when the field is associated with a plug-in element or event handler.

Return Values

- **IFX_SUCCESS**
This function always succeeds.

Related Topics

[IFXM - ProcessElementStylusDragEvent](#)

[IFXM - ProcessElementStylusDownEvent](#)

[IFXM - ProcessElementStylusUpEvent](#)

[IFXM - CreateElement](#)

IFXM - ProcessElementStylusDownEvent

This function is called when a stylus down event is detected by Inflexion UI Runtime that intersects with the element's bounding rectangle on screen.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_ProcessElementStylusDownEvent_<link  
    prefix> (  
    IFX_HMODULE      hModule,  
    IFX_HELEMENT     hElement,  
    IFX_INT32        x,  
    IFX_INT32        y,  
    IFX_INT32*       pConsumed);
```

Parameters

- **hModule**
The module handle returned by IFXM - Initialize.
- **hElement**
The element's handle for which data is being requested. It is returned by IFXM - CreateElement when the field is associated with a plug-in element or event handler.
- **x**
The x coordinate of the stylus event.
- **y**
The y coordinate of the stylus event.
- **pConsumed**
Returns an IFX_INT32 indicating whether the event is consumed. 1 means it was consumed, 0 means it was not.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR**
Any error. The Inflexion UI Runtime processes the event as though it was not consumed.

Description

The module should inspect the intersection coordinates and decide whether this element should grab the stylus focus (and thus be sole recipient of any subsequent stylus drag or stylus up events) or not. If focus is not grabbed (by filling in the pConsumed parameter by 1), then this

element may still get further stylus drag or stylus up events depending on whether something else in the UI grabs the stylus focus or not. In any case, a “stylus cancel” event will be sent to the plug-in element to indicate that the element should no longer expect any stylus events (until the next stylus down event, of course).

The given coordinate is a pixel position from the top-left corner of the plug-in element. 0,0 is assumed to be the top-left corner, with positive x and y axis extending right and down, respectively.

Related Topics

[IFXM - ProcessElementStylusDragEvent](#)

[IFXM - ProcessElementStylusUpEvent](#)

[IFXM - ProcessElementStylusCancelEvent](#)

IFXM - ProcessElementStylusUpEvent

This function is called when a stylus up event is detected by the Inflexion UI Runtime. The stylus up event will be offered to each Inflexion UI Runtime element in turn until one consumes it.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_ProcessElementStylusUpEvent_<link  
prefix> (  
    IFX_HMODULE hModule,  
    IFX_HELEMENT hElement,  
    INT32 x,  
    INT32 y);
```

Parameters

- **hModule**
The module handle returned by IFXM - Initialize.
- **hElement**
The element's handle for which data is being requested. It is returned by IFXM_CreateElement when the field is associated with a plugin element or event handler.
- **x**
The x coordinate of the stylus event.
- **y**
The y coordinate of the stylus event.

Return Values

- **IFX_SUCCESS**
Element has consumed the “stylus up” event.
- **IFX_ERROR**
Element has not consumed the “stylus up” event.

Description

The order of precedence is based on the order in which the elements under the stylus event coordinates are drawn to the screen (the element “on top” is offered the event first). If an element grabs the stylus event focus on the “stylus down” event or on a subsequent “stylus drag” event, it (and it alone) is offered the “stylus up” event. If the module wants to consume this event, it should perform whatever action is required (such as asking for a link to be executed, for example) and return IFX_SUCCESS. If the module is not interested in this event, it should return IFX_ERROR.

The given coordinate is a pixel position from the top-left corner of the plug-in element. 0,0 is assumed to be the top-left corner, with positive x and y axis extending right and down, respectively.

IFXM - GetElementCaretPosition

This function is used by the Inflexion UI Runtime to get the selection caret position for a basic text element that currently has the focus.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_GetElementCaretPosition_<link prefix> (  
    IFX_HMODULE    hModule,  
    IFX_HELEMENT   hElement,  
    IFX_INT32*     pIndex);
```

Parameters

- **hModule**
The module handle returned by IFXM - Initialize.
- **hElement**
The element's handle for which data is being requested. It is returned by IFXM - CreateElement when the field is associated with an event handler.
- **pIndex**
Returns the index of the caret, or -1 for no caret.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR**
Any error. Inflexion UI Runtime does not display a caret.

Description

This function is used by Inflexion UI Runtime to get the selection caret position for a basic text element that currently has the focus. The index returned refers to the zero-based indexes of the gaps between the characters. An index of zero indicates that the caret should be placed before the first character, and an index of 1 places the caret after the first character. An index of -1 indicates that the caret should be omitted. If the caret position is off the end of the string, it is positioned at the end of the string.

The function is called when Inflexion UI Runtime receives a field update on a text element that is focused and has an associated event handler. It also calls it when the text element first receives the focus. This function is only be called for an IFX_MODE_EVENT_HANDLER element.

Example

```
IFX_RETURN_STATUS
IFXM_ifxuisimpleedit_GetElementCaretPosition_simpleEdit(
    IFX_HMODULE      hModule,
    IFX_HELEMENT     hElement,
    IFX_INT32        *pIndex)
{
    SIMPLE_EDIT* pElementSession = (SIMPLE_EDIT*)hElement;

    if (pElementSession)
    {
        *pIndex = pElementSession->caretPosition;

        return IFX_SUCCESS;
    }

    return IFX_ERROR;
}
```

Related Topics

[IFXM - PositionElement](#)

IFXM - PositionElement

The Inflexion UI Runtime calls this function on each plugin element each time the visual representation of the element changes.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_PositionElement_<link prefix> (  
    IFX_HMODULE      hModule,  
    IFX_HELEMENT     hElement,  
    IFX_POSITION*    pPosition);
```

Parameters

- **hModule**
The module handle returned by IFXM - Initialize.
- **hElement**
The element's handle for which data is being requested. It is returned by IFXM - CreateElement when the field is associated with a plugin element or event handler.
- **pPosition**
Pointer to a struct containing information about the element's position.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR**
Any error. Inflexion UI destroys the element using IFXM - DestroyElement and the element no longer appears on the screen. IFXM - DeactivateElement is not to be called first.

Description

Inflexion UI calls this function on each plugin element each time the “placement” (that is, any of the degrees of freedom of the element that affects the visual representation of the element on screen) of the element changes. All subsequent frames should be rendered to the new viewport. Reposition calls may happen multiple times in rapid succession, and for buffered elements there is no guarantee as to how many paint calls Inflexion UI Runtime makes between reposition calls. Only activated elements are repositioned, and Inflexion UI Runtime does not deactivate the element first. This function is not called for an IFX_MODE_EVENT_HANDLER element.

Note that negative position coordinates, or large element dimensions, may result in clipping on any of the four sides of the element. Also note that elements that do not use some components of the position (such as opacity/color) ignore changes in these parts.

This method passes in different information depending on the current mode of the element referenced by hElement. Direct mode elements are notified each time the client area on the

screen changes. Therefore, they must inspect the left, top, width and height members of the position structure. Buffer mode elements must ignore the left, top, width and height parameters, because they are never set for elements in this mode.

```
typedef struct tIFX_Position
{
    INT32 left;
    INT32 top;
    INT32 width;
    INT32 height;
    INT32 opacity;
    INT32 color;
    INT32 frame;
} IFX_POSITION;
```

- left, top – integers in the range (-infinity:infinity), representing the location of the top left of the element relative to the top-left-hand corner of the screen, with positive numbers being pixels to the right and downward from this origin.
- width, height – integers in the range (-infinity:infinity), representing the desired viewport dimensions. Any number less than or equal to 0 indicate that the plugin should not attempt to paint the element to screen.
- opacity – a value between 0 and 100 indicating the opacity of the element, with 0 indicating full transparency and 100 indicating fully opaque.
- color – 32 bit value indicating the RGBA color of the element, with the least significant byte indicating the alpha channel component, and the most significant byte indicating the red channel component.
- frame – integer indicating which frame should now be shown.

Example

```
IFX_RETURN_STATUS IFXM_ifxuidemo_PositionElement_Direct (
    IFX_HMODULE      hModule,
    IFX_HELEMENT     hElement,
    IFX_POSITION     *pPosition)
{
    rect rect1;

    rect1.Xmin = pPosition->left;
    rect1.Xmax = pPosition->left + pPosition->width;
    rect1.Ymin = pPosition->top;
    rect1.Ymax = pPosition->top + pPosition->height;

    RS_Rectangle_Draw(FILL, &rect1, -1,0,0);

    return IFX_SUCCESS;
}
```

Related Topics

[IFXM - DestroyElement](#)

[IFXM - DeactivateElement](#)

IFXM - PaintElement

This function is called by Inflexion UI on each active buffered mode element when the Inflexion UI Runtime is ready to re-render the element.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_PaintElement_<link prefix> (  
    IFX_HMODULE    hModule,  
    IFX_HELEMENT   hElement);
```

Parameters

- **hModule**
The module handle returned by IFXM - Initialize.
- **hElement**
The element's handle for which data is being requested. It is returned by IFXM - CreateElement when the field is associated with a plug-in element or event handler.
- **IfxRenderContext**
Allows the module to pass any data (such as the Texture ID, for example) to Inflexion. Java modules only.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR**
Any error. Inflexion UI destroys the element using IFXM - DestroyElement and the element no longer appears on the screen. IFXM - DeactivateElement is not to be called first.

Description

This function is called periodically by Inflexion UI Runtime on each active buffered mode element to indicate that Inflexion UI is about to re-render the element. All painting should be completed before the function returns. Painting should be as efficient as possible. It is not necessary to paint to every pixel within the bounding box of the element, and the implementation may blend its pixels with the color values already in the buffer if desired.

This function is not called for event handler elements.

Calls to IFXM - PaintElement is only made in between activation and deactivation of the specified element.

Example

```
IFX_RETURN_STATUS IFXM_ifxuidemo_PaintElement_Buffered(
```

```
        IFX_HMODULE      hModule,  
        IFX_HELEMENT    hElement)  
{  
    int i, j;  
    for(i=0; i<context->height; i++)  
    {  
        for(j=0; j<context->width; j++)  
        {  
            context->pBuffer[i*context->width*4 + j*4] = 0xFF; /*R*/  
            context->pBuffer[i*context->width*4 + j*4 + 1] = 0; /*G*/  
            context->pBuffer[i*context->width*4 + j*4 + 2] = 0; /*B*/  
            context->pBuffer[i*context->width*4 + j*4 + 3] = 0xFF; /*A*/  
        }  
    }  
  
    return IFX_SUCCESS;  
}
```

Related Topics

[IFXM - CreateElement](#)

[IFXM - PositionElement](#)

IFXI_RequestRefreshBufferedElement

This function can be called at any time, either in the Inflexion UI Runtime execution context or from a different context. It notifies the Inflexion UI Runtime that the element must be repainted on the next paint cycle. If the element referenced by element is “active”, then Inflexion UI Runtime issues a IFXM_PaintElement call on this element. If the element is not in a buffered mode, or if it is deactivated, this event is ignored. This function can succeed if another module has exclusivity. Note that if the module updates the buffer and does not notify the Inflexion UI Runtime by using this method, the behavior is undefined (the changes may or may not be drawn to the screen).

Usage

```
IFX_RETURN_STATUS IFXI_RequestRefreshBufferedElement (
    IFX_HMODULEID hModuleId,
    IFX_HELEMENTID hElementId);
```

Parameters

- **hModuleId**
The handle supplied to the module by IFXM - Initialize.
- **hElementId**
The element identifier supplied by IFXM - CreateElement.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR_ASYNC_BLOCKING**
The Inflexion UI Runtime is currently executing an asynchronous link and cannot process the request at this time.
- **IFX_ERROR_INVALID_MODULEID**
The hModuleID parameter does not match an initialized module.
- **IFX_ERROR_ENGINE_NOT_READY**
The runtime is in the process of restarting following a critical error.

Related Topics

IFXI_RequestResizeBufferedElementBuffer	IFXM - Initialize
IFXM - CreateElement	IFXM - PaintElement

IFXI_RequestResizeBufferedElementBuffer

This function can be called at any time, either in the Inflexion UI Runtime execution context or from a different context. Inflexion UI is notified that the element needs its associated buffer resized.

Inflexion UI Runtime attempts to create the new buffer and, if successful, replaces and destroys the old buffer in the current rendering context structure. This is why it is vital that the module store the pointer to the context structure passed in the IFXM - ChangeElementMode call, rather than copying the structure. This function can succeed if another module has exclusivity.

Usage

```
IFX_RETURN_STATUS IFXI_RequestResizeBufferedElementBuffer (
    IFX_HMODULEID hModuleID,
    IFX_HELEMENTID hElementId,
    IFX_INT32 x,
    IFX_INT32 y);
```

Parameters

- **hModuleId**
The handle supplied to the module by IFXM - Initialize.
- **hElementId**
The element identifier supplied by IFXM - CreateElement.
- **x**
The buffer's width in pixels.
- **y**
The buffer's height in pixels.

Return Values

- **IFX_SUCCESS**
The buffer is allocated. The module is free to use the buffer immediately.
- **IFX_ERROR**
The buffer could not be allocated.
- **IFX_ERROR_ASYNC_BLOCKING**
The Inflexion UI Runtime is currently executing an asynchronous link and cannot process the request at this time.
- **IFX_ERROR_INVALID_MODULEID**
The hModuleID parameter does not match an initialized module.
- **IFX_ERROR_ENGINE_NOT_READY**
The runtime is in the process of restarting following a critical error.

Related Topics

[IFXI_RequestRefreshBufferedElement](#)

IFXI_RequestFullScreenStart

This function can only be called from IFXM - ExecuteLink and requests that Inflexion UI Runtime allow the specified plugin element to draw to the whole screen. The Inflexion UI Runtime stops painting other parts of the UI and deactivates all other plugins. The Inflexion UI Runtime repositions the element to fill the Inflexion UI Runtime client area. This function can succeed if another module has exclusivity.

Usage

```
IFX_RETURN_STATUS IFXI_RequestFullScreenStart (  
    IFX_HMODULEID hModuleID,  
    IFX_HELEMENTID hElementID);
```

Parameters

- **hModuleId**
The handle supplied to the module by IFXM - Initialize.
- **hElementId**
The element identifier supplied by IFXM - CreateElement.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR**
The element handle is invalid, or the element cannot occupy the full screen. This can occur because another screen is already occupied.
- **IFX_ERROR_ASYNC_BLOCKING**
The Inflexion UI Runtime is currently executing an asynchronous link and cannot process the request at this time.
- **IFX_ERROR_INVALID_MODULEID**
The hModuleID parameter does not match an initialized module.
- **IFX_ERROR_ENGINE_NOT_READY**
The runtime is in the process of restarting following a critical error.

Related Topics

[IFXI_RequestFullScreenStop](#)

[IFXM - ExecuteLink](#)

IFXI_RequestFullScreenStop

This function can be called at any time, either in the Inflexion UI Runtime execution context or from a different context. It requests that the Inflexion UI Runtime stop displaying in full screen mode, revert the current full screen plugin to normal, reactivate any deactivated plugins as required, and resume the paint cycle. The function is ignored if the Inflexion UI Runtime does not show the specified element full screen. The plugin is in full screen mode until it receives a call to IFXM - PositionElement returning it to its pre-full screen mode position. This function can succeed if another module has exclusivity.

Usage

```
IFX_RETURN_STATUS IFXI_RequestFullScreenStop (
    IFX_HMODULEID hModuleID,
    IFX_HELEMENTID hElementId);
```

Parameters

- **hModuleId**
The handle supplied to the module by IFXM - Initialize.
- **hElementId**
The element identifier supplied by IFXM - CreateElement.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR_ASYNC_BLOCKING**
The Inflexion UI Runtime is currently executing an asynchronous link and cannot process the request at this time.
- **IFX_ERROR_INVALID_MODULEID**
The hModuleID parameter does not match an initialized module.
- **IFX_ERROR_ENGINE_NOT_READY**
The runtime is in the process of restarting following a critical error.

Related Topics

[IFXI_RequestFullScreenStart](#)

Functions

The following method template supports the execution of links that a module has defined and a theme has used.

- [IFXM - ExecuteLink](#)
- [IFXE_Async_Link_Complete](#)
- [IFXI_RequestExecuteLink](#)
- [IFXI_RequestTriggerKey](#)

IFXM - ExecuteLink

This function is called when you click a specific link type served by this module. It executes the link using the <link params> parameters. Note that if the link type is declared as “functionAsync” in the module definition file, Inflexion UI Runtime is in the asynchronous link suspend state until notified through the IFXE_Async_Link_Complete method that the link is complete.



Note

IFXE_Async_Link_Complete should not be called within the context of the IFXM - ExecuteLink call; IFX_Error should be returned instead. This prevents the Runtime component from going into the asynchronous link state.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_ExecuteLink_<link prefix> (  
    IFX_HMODULE  hModule,  
    <link params>);
```

Parameters

- hModule
The module handle returned by IFXM - Initialize.
- <link params>
A sequence of parameters matching the body and the list of link parameters defined in the module definition.

Return Values

- IFX_SUCCESS
Successful completion.
- IFX_ERROR
Any error. Inflexion UI Runtime does nothing, as if the clicked link is inactive.

Example

```
IFX_RETURN_STATUS  
IFXM_ifxuisimpleedit_ExecuteLink_cycleInputMode(IFX_HMODULE  hModule)  
{  
    SIMPLE_EDIT_SESSION* pLocalSession = (SIMPLE_EDIT_SESSION*)hModule;  
  
    if (pLocalSession)  
    {  
        if (pLocalSession->pCurrentElement)  
        {  
            /* Cycle the input mode on the focused element */  
            SimpleEdit_CycleInputMode(pLocalSession, pLocalSession->pCurrentElement);  
        }  
    }  
}
```

```
        }  
    }  
  
    return IFX_SUCCESS;  
}
```

Related Topics

[IFXE_Async_Link_Complete](#)

IFXE_Async_Link_Complete

This function causes Inflexion UI Runtime to animate back from the asynchronous link launch state. This causes the whole screen to be repainted.

Calling this function when not in the asynchronous link launch state has no effect.

Usage

```
IFX_RETURN_STATUS IFXE_Async_Link_Complete (void);
```

Return Values

- **IFX_SUCCESS**
The command was successful.
- **IFX_ERROR_ENGINE_NOT_READY**
The runtime is in the process of restarting following a critical error.
- **IFX_ERROR**
Any other error.

Related Topics

[IFXE_Refresh_Display](#)

IFXI_RequestExecuteLink

This function can be called at any time, either in the Inflexion UI execution context or from a different context. This function causes Inflexion UI Runtime to execute the given executable link. Link execution is subject to the exclusivity rules.



Note

Navigation-type links (those with *node://* prefix) are not permitted. Modules can only request navigation through the *entrypoint_id* part of the IFXE_CONFIGURATION structure and the IFXE API.

Usage

```
IFX_RETURN_STATUS IFXI_RequestExecuteLink (  
    IFX_HMODULEID hModuleID,  
    const IFX_WCHAR* szUri);
```

Parameters

- **hModuleId**
The handle supplied to the module by IFXM - Initialize.
- **szUri**
The link to execute.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR**
Any error. The link is not executed. A handle might be invalid.
- **IFX_ERROR_EXCLUSIVITY**
Returned if the link cannot be executed because another module has exclusivity.
- **IFX_ERROR_ASYNC_BLOCKING**
The Inflexion UI Runtime is currently executing an asynchronous link and cannot process the request at this time.
- **IFX_ERROR_INVALID_MODULEID**
The hModuleID parameter does not match an initialized module.
- **IFX_ERROR_ENGINE_NOT_READY**
The runtime is in the process of restarting following a critical error.

IFXI_RequestTriggerKey

This function can be called at any time, either in the Inflexion UI execution context or from a different context. This function causes the Inflexion UI Runtime to simulate the given key trigger. Triggering keys is subject to the exclusivity rules.

Usage

```
IFX_RETURN_STATUS IFXI_RequestTriggerKey (  
    IFX_HMODULEID hModuleID,  
    IFX_INT32 scancode);
```

Parameters

- **hModuleId**
The handle supplied to the module by IFXM - Initialize.
- **scancode**
The scancode of the key to trigger.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR**
Any error. The trigger is not executed. A handle might be invalid.
- **IFX_ERROR_EXCLUSIVITY**
Returned if the trigger cannot be executed because another module has exclusivity.
- **IFX_ERROR_ASYNC_BLOCKING**
The Inflexion UI Runtime is currently executing an asynchronous link and cannot process the request at this time.

Exclusivity

The following method and the callback template allow modules to negotiate exclusive access to Inflexion UI:

- [IFXM - ExclusivityStatusChange](#)
- [IFXI_ReleaseExclusivity](#)
- [IFXI_ReleaseExclusivity](#)

IFXM - ExclusivityStatusChange

This function is called when the module's exclusivity status changes.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_ExclusivityStatusChange (  
    IFX_HMODULE          hModule,  
    IFX_EXCLUSIVITY_STATUS status);
```

Parameters

- hModule

The module handle returned by IFXM - Initialize.

- status

New status that can be one of the following:

IFX_EXCLUSIVITY_STATUS_GRANTED

A previous request for exclusivity is granted.

IFX_EXCLUSIVITY_STATUS_DENIED

A previous request for exclusivity is denied because the request has timed out.

IFX_EXCLUSIVITY_STATUS_SUSPENDED

The module had exclusivity, but exclusivity is transferred to a module requesting it with a higher priority.

IFX_EXCLUSIVITY_STATUS_RESUMED

The module previously had exclusivity suspended, but it regained exclusivity.

IFX_EXCLUSIVITY_STATUS_ERROR

An internal error within the exclusivity system occurred. The module does not have exclusivity.

Return Values

- IFX_SUCCESS

Successful completion.

IFXI_ReleaseExclusivity

The module calls this function to release exclusivity. Note that this method is only generated if the module has declared that it supports exclusivity in the module definition file, and that each exclusivity-enabled module has its own version.

Usage

```
IFX_RETURN_STATUS IFXI_<module name>_ReleaseExclusivity (  
    IFX_HMODULEID hModuleID);
```

Parameters

- **hModuleId**
The handle supplied to the module by IFXM - Initialize.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR_INVALID_MODULEID**
The hModuleID parameter does not match an initialized module.

Related Topics

[IFXI_RequestExclusivity](#)

IFXI_RequestExclusivity

The module calls this function to request exclusive access to Inflexion UI Runtime and the UI state machine. Note that this method is only generated if the module has declared that it supports exclusivity in the module definition file, and that each exclusivity-enabled module has its own version.

Usage

```
IFX_RETURN_STATUS IFXI_<module name>_RequestExclusivity (  
    IFX_HMODULEID hModuleID,  
    IFX_INT32 priority,  
    IFX_INT32 timeout);
```

Parameters

- **hModuleId**
The handle supplied to the module by IFXM - Initialize.
- **priority**
An integer representing the priority of the module. Can be 0 to 31 inclusive, with 0 being the highest priority.
- **timeout**
An integer defining the length of time in milliseconds to wait to gain exclusivity. IFX_WAIT_INFINITE can be used to wait indefinitely.

Return Values

- **IFX_SUCCESS**
Exclusivity was granted, if timeout is 0. Otherwise, the request is added to the queue, and further information is given to the module via IFXM - ExclusivityStatusChange.
- **IFX_ERROR_EXCLUSIVITY**
Only returned if timeout is 0 and exclusivity could not be granted.
- **IFX_ERROR**
Any other error.
- **IFX_ERROR_INVALID_MODULEID**
The hModuleID parameter does not match an initialized module.

Related Topics

[IFXI_ReleaseExclusivity](#)

[IFXM - ExclusivityStatusChange](#)

Shared Buffers

The following methods and method templates allow buffer sharing between modules:

- [IFXM - BufferCreateString](#)
- [IFXM - BufferDestroy](#)
- [IFXM - BufferSetFocus](#)
- [IFXM - BufferUnsetFocus](#)
- [IFXI_BufferCreateString](#)
- [IFXI_BufferDestroy](#)
- [IFXI_BufferSetFocus](#)
- [IFXI_BufferUnsetFocus](#)

IFXM - BufferCreateString

The Integration Layer calls this function when another module requests access to an editable field buffer. A module is only asked to implement this function if the field is declared as an editable string in the module definition file. If the field is associated with a menu in the module definition file, then the <link prefix> version is used.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_BufferCreateString_<link prefix>_<field
name> (
    IFX_HMODULE hModule,
    IFX_HBUFFER* phBuffer,
    IFX_HMENU hMenu,
    IFX_INT32 item,
    IFX_WCHAR** pBuffer,
    IFX_INT32 maxLength);
```

```
IFX_RETURN_STATUS IFXM_<module name>_BufferCreateString_<field name> (
    IFX_HMODULE hModule,
    IFX_HBUFFER* phBuffer,
    IFX_HMENU hMenu,
    IFX_INT32 item,
    IFX_WCHAR** pBuffer,
    IFX_INT32 maxLength);
```

Parameters

- **hModule**
The module handle returned by IFXM - Initialize.
- **phBuffer**
Returns the buffer session handle allocated by the module to use when the Integration Layer returns to the module regarding this buffer.
- **hMenu**
The menu's handle for which data is being requested. It is returned by IFXM - OpenMenu if the data is associated with a menu.
- **item**
The index (0-based) of the menu item for which data is requested if the data is associated with an item. Otherwise -1.

- **pBuffer**
Returns a pointer to a buffer that the element can edit. It is pre-populated with the field data.
- **maxLength**
The maximum number of characters that the buffer can contain, excluding the NULL terminator.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR**
Any error. IFXM - BufferDestroy is not called, therefore all memory must be freed.

Example

```
IFX_RETURN_STATUS IFXM_ifxuidemo_BufferCreateString_menuPlugin_caption(  
    IFX_HMODULE          hModule,  
    IFX_HBUFFER          *pBuffer,  
    IFX_HMENU            hMenu,  
    IFX_INT32            item,  
    IFX_WCHAR            **pBuffer,  
    IFX_INT32            maxLength)  
{  
    CACHE_MENU* pMenu = (CACHE_MENU*)hMenu;  
  
    if (NU_SUCCESS == NU_Allocate_Memory(  
        &System_Memory,  
        (void **)pBuffer,  
        (maxLength + 1) * sizeof( IFX_WCHAR ),  
        NU_NO_SUSPEND))  
    {  
        memset(*pBuffer, 0, (maxLength + 1) * sizeof( IFX_WCHAR ));  
  
        /* Populate the buffer with the default caption string */  
        lc_wcsncpy(*pBuffer, pMenu->menuData[item].captionAttr, maxLength);  
        *pBuffer = (IFX_HBUFFER)*pBuffer;  
  
        return IFX_SUCCESS;  
    }  
  
    return IFX_ERROR;  
}
```

Related Topics

[IFXM - BufferDestroy](#)

[IFXM - BufferSetFocus](#)

[IFXM - BufferUnsetFocus](#)

IFXM - BufferDestroy

The Integration Layer calls this function when another module is finished editing the field buffer previously acquired by a call to IFXM - BufferCreateString. A module is only asked to implement this function if it declares the field as an editable string in the module definition file. If the field is associated with a menu in the module definition file, then the <link prefix> version is used.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_BufferDestroy_<link prefix>_<field
name> (
    IFX_HMODULE      hModule,
    IFX_HBUFFER      hBuffer);

IFX_RETURN_STATUS IFXM_<module name>_BufferDestroy_<field name> (
    IFX_HMODULE      hModule,
    IFX_HBUFFER      hBuffer);
```

Parameters

- **hModule**
The module handle returned by IFXM - Initialize.
- **hBuffer**
The buffer session handle allocated by the module to use when the Integration Layer returns to the module regarding this buffer.

Return Values

- **IFX_SUCCESS**
Successful completion.

Example

```
IFX_RETURN_STATUS
IFXM_ifxuidemo_BufferDestroy_menuPlugin_caption(IFX_HMODULE  hModule,
                                                IFX_HBUFFER  hBuffer)
{
    if (hBuffer)
        NU_Deallocate_Memory(hBuffer);

    return IFX_SUCCESS;
}
```

Related Topics

[IFXM - BufferCreateString](#)

[IFXM - BufferSetFocus](#)

[IFXM - BufferUnsetFocus](#)

IFXM - BufferSetFocus

The Integration Layer calls this function when another module notifies the Integration Layer that the element containing an editable field buffer belonging to this module has the focus. A module is only asked to implement this function if it declares the field as an editable string in the module definition file. If the field is associated with a menu in the module definition file, then the <link prefix> version is used.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_BufferSetFocus_<link prefix>_<field
name> (
    IFX_HMODULE      hModule,
    IFX_HBUFFER      hBuffer);

IFX_RETURN_STATUS IFXM_<module name>_BufferSetFocus_<field name> (
    IFX_HMODULE      hModule,
    IFX_HBUFFER      hBuffer);
```

Parameters

- **hModule**
The module handle returned by IFXM - Initialize.
- **hBuffer**
The buffer session handle allocated by the module to use when the Integration Layer returns to the module regarding this buffer.

Return Values

- **IFX_SUCCESS**
Successful completion.

Example

```
IFX_RETURN_STATUS IFXM_ifxuidemo_BufferSetFocus_menuPlugin_caption(
    IFX_HMODULE      hModule,
    IFX_HBUFFER      hBuffer)
{
    return IFX_SUCCESS;
}
```

Related Topics

[IFXM - BufferCreateString](#)[IFXM - BufferDestroy](#)[IFXM - BufferUnsetFocus](#)

IFXM - BufferUnsetFocus

The Integration Layer calls this function to inform the module that the element associated with the buffer has the focus removed. A module is only asked to implement this function if it declares the field as an editable string in the module definition file. If the field is associated with a menu in the module definition file, then the <link prefix> version is used.

Usage

```
IFX_RETURN_STATUS IFXM_<module name>_BufferUnsetFocus_<link prefix>_<field
name> (
    IFX_HMODULE      hModule,
    IFX_HBUFFER      hBuffer);
```

```
IFX_RETURN_STATUS IFXM_<module name>_BufferUnsetFocus_<field name> (
    IFX_HMODULE      hModule,
    IFX_HBUFFER      hBuffer);
```

Parameters

- **hModule**
The module handle returned by IFXM - Initialize.
- **hBuffer**
The buffer session handle allocated by the module to use when the Integration Layer returns to the module regarding this buffer.

Return Values

- **IFX_SUCCESS**
Successful completion.

Example

```
IFX_RETURN_STATUS IFXM_ifxuidemo_BufferUnsetFocus_menuPlugin_caption(
    IFX_HMODULE hModule,
    IFX_HBUFFER hBuffer)
{
    return IFX_SUCCESS;
}
```

Related Topics

[IFXM - BufferCreateString](#)

[IFXM - BufferDestroy](#)

[IFXM - BufferSetFocus](#)

IFXI_BufferCreateString

The module calls this function during a IFXM - CreateElement call to request a buffer from another module, allowing edition of the field data owned by the other module. The Integration Layer forwards the request to the relevant module using the IFXM - BufferCreateString call. Only fields defined in the module definition file with the editType attribute set to 'string' can be accessed in this way by another module. "Global" field names are unique in a system, and they can be accessed from any module. Menu-context field data requires the corresponding menu ID. Therefore, a module element can only access menu field data buffers for the current page.

Usage

```
IFX_RETURN_STATUS IFXI_BufferCreateString (
    IFX_HMODULEID hModuleId,
    IFX_HBUFFERID* phBufferId,
    IFX_HMENUID hMenuId,
    IFX_INT32 item,
    const IFX_WCHAR* szField,
    IFX_WCHAR** pBuffer,
    IFX_INT32 maxLength);
```

Parameters

- **hModuleId**
The handle supplied to the module by IFXM - Initialize.
- **phBufferId**
Returns the buffer identifier to use when the element returns to the Integration Layer regarding this buffer.
- **hMenuId**
The menu identifier to use when the module returns to the Integration Layer regarding the supplied menu context.
- **item**
The index (0-based) of the menu item for which data is requested if the data is associated with an item. Otherwise -1.
- **szField**
The field name for which data is requested (for example, "artist" or "title").
- **pBuffer**
Returns a pointer to a buffer that the element can edit. It is pre-populated with the field data.
- **maxLength**
The maximum number of characters the buffer can contain, excluding the NULL terminator.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR**
Any error (other than **IFX_ERROR_INVALID_MODULEID**). **IFXI - BufferDestroy** is not called, therefore all memory must be freed.
- **IFX_ERROR_INVALID_MODULEID**
The **hModuleID** parameter does not match an initialized module.

Related Topics

[IFXI_BufferDestroy](#)

[IFXM - CreateElement](#)

IFXI_BufferDestroy

The module calls this function during a IFXM - DestroyElement call to free the buffer allocated by IFXM - BufferCreateString. The Integration Layer forwards the request to the relevant module using the IFXM - BufferDestroy calls.

Usage

```
IFX_RETURN_STATUS IFXI_BufferDestroy (  
    IFX_HMODULEID hModuleID,  
    IFX_HBUFFERID hBufferId);
```

Parameters

- **hModuleId**
The handle supplied to the module by IFXM - Initialize.
- **hBufferId**
The buffer identifier to use when the element returns to the Integration Layer regarding this buffer.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR**
Buffer not recognized.
- **IFX_ERROR_INVALID_MODULEID**
The hModuleID parameter does not match an initialized module.

Related Topics

[IFXI_BufferCreateString](#)

[IFXM - DestroyElement](#)

IFXI_BufferSetFocus

The module calls this function during a IFXM - SetElementFocus call to inform the module that the element associated with the buffer is the focus. The Integration Layer forwards the request to the relevant module using the IFXM - BufferSetFocus call.

Usage

```
IFX_RETURN_STATUS IFXI_BufferSetFocus (  
    IFX_HMODULEID hModuleID,  
    IFX_HBUFFERID hBufferId);
```

Parameters

- **hModuleId**
The handle supplied to the module by IFXM - Initialize.
- **hBufferId**
The buffer identifier to use when the element returns to the Integration Layer regarding this buffer.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR**
Buffer not recognized.
- **IFX_ERROR_INVALID_MODULEID**
The hModuleID parameter does not match an initialized module.

Related Topics

[IFXI_BufferUnsetFocus](#)

[IFXM - SetElementFocus](#)

IFXI_BufferUnsetFocus

The module calls this function during a IFXM - UnsetElementFocus call to inform the module owning the editable field buffer that the element associated with the buffer is no longer the focus. The Integration Layer forwards the request to the relevant module using the IFXM - BufferUnsetFocus call.

Usage

```
IFX_RETURN_STATUS IFXI_BufferUnsetFocus (
    IFX_HMODULEID hModuleID,
    IFX_HBUFFERID hBufferId);
```

Parameters

- **hModuleId**
The handle supplied to the module by IFXM - Initialize.
- **hBufferId**
The buffer identifier to use when the element returns to the Integration Layer regarding this buffer.

Return Values

- **IFX_SUCCESS**
Successful completion.
- **IFX_ERROR**
Buffer not recognized.
- **IFX_ERROR_INVALID_MODULEID**
The hModuleID parameter does not match an initialized module.

Related Topics

[IFXI_BufferSetFocus](#)

[IFXM - UnsetElementFocus](#)

Chapter 10

Porting API Reference

This chapter details the Porting API, which provides Inflexion UI with methods to manipulate canvasses (bitmap buffers), native fonts and displays, and to access a file system.

All of the porting API functions and specific error codes are listed in the header file, *nucleus/os/ui/inflexionui/engine/ifxui_porting.h*.

In this chapter:

About Type Definitions	223
Initialization and Shutdown	225
Display Configuration	229
Canvas Management	234
Custom Image Format Unpacking	239
Rendering	246
Native Fonts	258
Error Notes	267
FILE Functionality	269
Timer Management	282
Debug Support	287
Memory Management	289
Mutexes	293
Other Methods	298
Validating a New Porting Layer	300

About Type Definitions

All type definitions specific to the porting API are in a single header, *nucleus/os/ui/inflexionui/engine/ifxui_porting.h*.

Table 10-1. Type Definitions

Type Definition	Description
IFXP_FONT_FACE	This enumeration determines the font face when creating a font canvas.

Table 10-1. Type Definitions (cont.)

Type Definition	Description
IFXP_TEXT	This structure contains the dimensions and alpha buffer that describe and hold a text canvas. It also contains an internal pointer that is implementation-specific.
IFXP_RECT	This structure contains the top, left, and bottom right x,y coordinates that describe a rectangle used by the rendering functions.
IFXP_FGBG	This enumeration determines whether a canvas is created as a foreground or background canvas.
IFXP_FONT	Generic pointer to a font object
IFXP_FILE	Generic pointer to a file object
IFXP_SEEK	This enumeration gives the possible set of seek attributes for the porting layer file system support.
IFXP_INFO	This enumeration lists the possible file object types.
IFXP_SEARCH	This structure contains the cached information used when performing a search using the porting layer file system support.
IFXP_TIMER	This is an implementation specific type cast to a void *. It identifies a timer initialized by the timer module
IFXP_MUTEX	This is an implementation specific type cast to a void *. It identifies a mutex initialized by the mutex module
IFXP_MEM_USE	This is an enumeration. It specifies the memory pool to be used when calling the memory management functions.
IFXP_CMD	This enumeration lists the commands supported by IFXP_Command.

Initialization and Shutdown

The following functions are called before any of the Font or Render functions and initialize and shut down any components required by those functions:

- [IFXP_Initialize](#)
- [IFXP_Shutdown](#)
- [IFXP_Runtime_State_Changed](#)

IFXP_Initialize

This function cannot be assumed to be called within the Inflexion UI execution context. It is called from the application context. If a specific platform has context-sensitive initialization to perform, it should be performed in IFXP_Change_Context.

This function initializes the sections of the Porting Layer relating to Fonts, Rendering, and Error Notes allocating resources where required.

Usage

```
IFX_RETURN_STATUS IFXP_Initialize (void);
```

Return Values

- IFX_SUCCESS
- IFX_ERROR
Any error.

Related Topics

[IFXP_Shutdown](#)

IFXP_Shutdown

This function frees any resources allocated in the initialize function and shuts down the sections of the Porting Layer relating to Fonts, Rendering, and Error Notes.

Usage

```
IFX_RETURN_STATUS IFXP_Shutdown (void);
```

Return Values

- IFX_SUCCESS
- IFX_ERROR

Any error.

Related Topics

[IFXP_Initialize](#)

IFXP_Runtime_State_Changed

This function is called each time the runtime state is changed, and may be used to broadcast relevant notification messages to the rest of the system if required. The parameter value will be one of the members of the IFXE_STATE enum.

Usage

```
IFX_RETURN_STATUS IFXP_Runtime_State_Changed (IFXE_STATE state);
```

Parameters

- state - New state of the runtime. One of:

IFXE_STATE_UNINITIALIZED

Initial and terminal state of the runtime.

IFXE_STATE_INITIALIZED

Indicates that the runtime has been successfully initialized by the framework.

IFXE_STATE_ACTIVE

Indicates that there is a theme loaded, and that the runtime is prepared to receive user input and/or module requests.

IFXE_STATE_INACTIVE_SUSPENDED

Indicates that the runtime has been placed into the suspended state by the framework. Module requests will be rejected.

IFXE_STATE_INACTIVE_ASYNC_LINK

Indicates that the runtime has been placed into the “asynchronous link” state by a module functional link being executed. Module requests and user input will be rejected until the IFXE_Async_Link_Complete method is called.

IFXE_STATE_INACTIVE_RESTARTING

Indicates that the runtime is recovering from a serious error (usually related to “Out of Memory” events), and cannot accept user input or module requests at this time.

Return Values

- IFX_SUCCESS

Related Topics

[IFXP_Initialize](#)

[IFXP_Shutdown](#)

[IFXE_Async_Link_Complete](#)

Display Configuration

The following functions are used to configure and retrieve information about the display structure:

- [IFXP_Display_Set_Mode](#)
- [IFXP_Display_Get_Info](#)

IFXP_Display_Set_Mode

This function populates the internal display configuration based on the supplied `display_mode`. This includes setting up any rotation parameters.

Usage

```
IFX_RETURN_STATUS IFXP_Display_Set_Mode (IFX_INT32 left, IFX_INT32 top,  
                                         IFX_INT32 width, IFX_INT32 height, const char* display_mode);
```

Parameters

- `left`
Specifies the x offset applied to a canvas when transferring to display.
- `top`
Specifies the y offset applied to a canvas when transferring to display.
- `width`
Specifies the desired output region width for displaying the theme.
- `height`
Specifies the desired output region height for displaying the theme.
- `display_mode`
The name of the display mode as defined in the platform definition file.

Return Values

- `IFX_SUCCESS`
- `IFX_ERROR`
Any error.

Description

The `IFX_DISPLAY` structure populated by this function is defined as follows:

```
typedef struct tagIFX_DISPLAY  
{  
    void *frameBuffer;  
    unsigned int width;  
    unsigned int height;  
    unsigned int stride;  
    IFX_ROTATE rotation;  
    unsigned int bpp;  
    unsigned int msaa_samples;  
    unsigned int depth_buffer_size;  
    unsigned int egl_swap_interval  
    int offsetX;  
    int offsetY;  
    int renderFull;  
    void *internal;
```

```
} IFX_DISPLAY;
```

- **frameBuffer**

frame_buffer is populated with a pointer to the display frame buffer. It may be in any color format. The direct plugin elements must match the configuration of this structure.

- **width and height**

Populated with the dimensions of the display.

- **stride**

Populated based on the length of a frameBuffer row in bytes.

- **rotation**

Populated with the IFX_ROTATE enum defined as follows:

```
typedef enum
{
    IFX_ROTATE_0 = 0,
    IFX_ROTATE_90,
    IFX_ROTATE_180,
    IFX_ROTATE_270
} IFX_ROTATE;
```

This is based on the display_mode string passed into this function.

- **bpp**

Defines the display color depth in bits per pixel. This is only consumed by the engine in OpenGL mode.

- **msaa_samples**

This defines the number of samples used for multi-sample anti-aliasing (MSAA). Setting this to 0 or 1 disables MSAA. This is only used by the engine in OpenGL mode.

- **depth_buffer_size**

Specifies the OpenGL ES depth buffer size. Defaults to 24.

- **egl_swap_interval**

Specifies the OpenGL ES swap interval. Defaults to 0.

- **offset_x**

Populated based on the left offset passed into this function.

- **offset_y**

Populated based on the top offset passed into this function.

- **render_full**

Set to 0 if this Porting Layer does not support full frame blitting every frame and nonzero if full frame blitting is supported.

- internal

This is provided to store a pointer to a structure describing the implementation's display engine. It is used by 'direct mode' plugin elements to allow them to prepare rendering contexts used to render the plugin content to the display in the plugin client window.

Related Topics

[IFXP_Display_Get_Info](#)

IFXP_Display_Get_Info

This function returns a pointer to the display information structure populated by IFXP_Display_Set_Mode.

Usage

```
IFX_RETURN_STATUS IFXP_Display_Get_Info (IFXP_DISPLAY** display);
```

Parameters

- display
The pointer to the populated display structure.

Return Values

- IFX_SUCCESS
- IFX_ERROR
Could not obtain display information.

Related Topics

[IFXP_Display_Set_Mode](#)

Canvas Management

The following functions are used to manage the creation and destruction of canvasses. Also included are two functions to manipulate rectangles on a canvas.

- [IFXP_Canvas_Create](#)
- [IFXP_Canvas_Destroy](#)
- [IFXP_Canvas_Draw_Rect_Fill](#)

IFXP_Canvas_Create

This function creates an IFX_CANVAS structure based on the IFX_CANVAS_MODE defined in *nucleus/os/ui/inflexionui/engine/.metadata*. IFXP_Canvas_Create allocates memory for a bitmap buffer. This may be by using a graphics library, or through standard allocation functions. For platforms that have tightly coupled memory, consider creating the canvas in this memory for better performance. Where layers are used, this function allocates or creates layers.

Usage

```
IFXP_RETURN_STATUS IFXP_Canvas_Create (
    IFXP_CANVAS** canvas,
    IFX_UINT32 width,
    IFX_UINT32 height,
    IFXP_FGBG fgbg);
```

Parameters

- **canvas**
Points to the pointer to the canvas structure to be filled with the address of the created canvas.
- **width**
Width in pixels of the canvas that must be created.
- **height**
Height in pixels of the canvas that must be created.
- **fgbg**
This may be either IFXP_FOREGROUND or IFXP_BACKGROUND. It specifies whether this is a foreground or a background canvas. This is only relevant on a platform that supports layer blending. It is up to the specific implementation to track the creation of background and foreground canvasses and destroy them appropriately.

Return Values

- IFX_SUCCESS
- IFX_ERROR
Any error.

Description

```
typedef struct tagIFX_DISPLAY
{
    void          *frame_buffer;
    IFX_UINT32 width;
    IFX_UINT32 height;
    IFX_UINT32 stride;
    IFX_ROTATE rotation;
    IFX_INT32  offset_x;
    IFX_INT32  offset_y;
```

```
    IFX_INT32  render_full;  
    Void      *internal;  
} IFX_DISPLAY;
```

The IFX_CANVAS structure is defined as follows:

```
typedef struct tagIFX_CANVAS  
{  
    void      *colorBuffer  
  
    IFX_UINT32 width;  
    IFX_UINT32 stride;  
    IFX_UINT32 height;  
    void      *internal;  
} IFX_CANVAS
```

- **color_buffer**

Must be populated with a pointer to the created bitmap buffer. This matches the color format defined in *nucleus/os/ui/inflexionui/engine/metadata* by the IFX_CANVAS_MODE_ define.

- **width and height**

Must be set to the width and height passed into this function.

- **stride**

Must be calculated based on the width and color format.

- **internal**

Is not used by the Inflexion UI Runtime and is provided for implementation-specific use, for example to store a pointer to a structure describing the implementation's graphic engine.

Related Topics

[IFXP_Canvas_Destroy](#)

[IFXP_Canvas_Draw_Rect_Fill](#)

IFXP_Canvas_Destroy

This function frees any resources previously allocated by IFXP_Canvas_Create.

Usage

```
IFX_RETURN_STATUS IFXP_Canvas_Destroy (IFXP_CANVAS *canvas);
```

Parameters

- **canvas**
Pointer to the canvas to destroy. If the specific implementation handles background or foreground canvasses differently, it is up to the implementation to handle this correctly.

Return Values

- **IFX_SUCCESS**
- **IFX_ERROR**
Any error

Related Topics

[IFXP_Canvas_Create](#)

[IFXP_Canvas_Draw_Rect_Fill](#)

IFXP_Canvas_Draw_Rect_Fill

This function draws a filled rectangle to the canvas.

Usage

```
IFX_RETURN_STATUS IFXP_Canvas_Draw_Rect_Fill (  
    IFXP_CANVAS* dst_canvas,  
    IFXP_RECT* dstRect,  
    IFX_INT32 color);
```

Parameters

- **dst_canvas**
This specifies the destination canvas to which the rectangle should be drawn.
- **dst_rect**
This specifies the rectangle to draw.
- **color**
Color of the rectangle.

Return Values

- **IFX_SUCCESS**
- **IFX_ERROR**
Any error.

Description

This function may be hardware accelerated on platforms that support it. IFXP_RECT is defined as follows:

```
typedef struct _ifxp_rect  
{  
    IFX_INT32 x1;  
    IFX_INT32 y1;  
    IFX_INT32 x2;  
    IFX_INT32 y2;  
} IFXP_RECT;
```

This is a rectangle described via its top left (x1,y1) and bottom right (x2,y2) vertices. This function may be hardware accelerated on platforms that support it.

Related Topics

[IFXP_Canvas_Create](#)

Custom Image Format Unpacking

The following functions support unpacking images packed using some format other than the formats supported by the Inflexion UI Runtime engine:

- [IFXP_Image_Open](#)
- [IFXP_Image_Get_Data](#)
- [IFXP_Image_Release_Data](#)
- [IFXP_Image_Close](#)
- [IFXP_Image_Check_Extension](#)

The image formats supported are dependent on the features enabled in the porting layer. See [Custom Image Format Support](#) for more information.

IFXP_Image_Open

This method should take the image file path and filename held in the `image_file` parameter, identify the file type from the file extension, open and read the file using the `IFXE_File_*` API, and gather sufficient information from the file to fill the `IFX_IMAGE_INFO` structure pointed to by the `info` parameter. This method should return `IFX_ERROR` if the file format is not recognized or is unsupported.

Usage

```
IFX_RETURN_STATUS IFXP_Image_Open(IFX_IMAGE_INFO* info, const char*  
    image_file);
```

Parameters

- `info`

Pointer to an `IFX_IMAGE_INFO` structure to be filled in with the image information. The structure fields are:

`type`

Type of image (`IFX_IMAGE_TYPE_RGBA_DATA` for normal image data, `IFX_IMAGE_TYPE_RGB_DATA` for background images or `IFX_IMAGE_TYPE_OPENGL_COMPRESSED_TEXTURE` for compressed image data for use in Open GL ES textures)

`width`

Width of the image in pixels

`height`

Height of the image in pixels

`length`

Number of bytes in the image data

`internal`

Pointer to a session structure for use by the `IFXP_Image_*` API implementation

`texture_format`

OpenGL ES enumeration value for the compressed texture format
`IFX_IMAGE_TYPE_OPENGL_COMPRESSED_TEXTURE` type files only.

`first_mipmap_level`

First mipmap level supported by the image file
`IFX_IMAGE_TYPE_OPENGL_COMPRESSED_TEXTURE` type files only.

`last_mipmap_level`

Last mipmap level supported by the image file
`IFX_IMAGE_TYPE_OPENGL_COMPRESSED_TEXTURE` type files only.

- **image_file**
Null-terminated string containing the path and file name (including file extension) of the image file to open. IFXE_File_Open should be used to open this file.

Return Values

- **IFX_SUCCESS**
Image file format unpacking is supported, and image unpacking session created.
- **IFX_ERROR**
Image file format unpacking is unsupported, or file is corrupt, or “out of memory” error.

Related Topics

[IFXP_Image_Get_Data](#)

[IFXP_Image_Release_Data](#)

[IFXP_Image_Close](#)

IFXP_Image_Get_Data

Create and populate a data buffer with image data from the unpacked image file, given a mip-map level. This will be typically called once for non-mipmapped images, and once per mipmap level for pre-mipmapped images.

Usage

```
IFX_RETURN_STATUS IFXP_Get_Data(IFX_IMAGE_INFO* info,  
    IFX_UINT32 level,  
    IFX_UINT32* length,  
    void** data);
```

Parameters

- **info**
Pointer to the IFX_IMAGE_INFO structure filled in during the call to IFXP_Image_Open.
- **level**
Required mip-map level for pre-mipmapped images of type IFX_IMAGE_TYPE_OPENGL_COMPRESSED_TEXTURE.
- **length**
Out parameter to be filled in with the size of the buffer pointed to by the data parameter.
- **data**
Out parameter to be filled with the address of a data buffer containing the image data. The format of the buffer will be suitable for the image type, and the size of the data buffer is contained in the length parameter. This buffer must be freed with a call to IFXP_Image_Release_Data when it is finished with.

Return Values

- **IFX_SUCCESS**
data parameter successfully filled with image data.
- **IFX_ERROR**
Any error.

Related Topics

[IFXP_Image_Open](#)

[IFXP_Image_Release_Data](#)

[IFXP_Image_Close](#)

IFXP_Image_Release_Data

Releases a data buffer allocated during a call to IFXP_Image_Get_Data.

Usage

```
IFX_RETURN_STATUS IFXP_Get_Data(IFX_IMAGE_INFO* info void* data);
```

Parameters

- **info**
Pointer to the IFX_IMAGE_INFO structure filled in during the call to IFXP_Image_Open.
- **data**
Address of a data buffer previously returned in a call to IFXP_Image_Get_Data.

Return Values

- **IFX_SUCCESS**
Function always succeeds

Related Topics

[IFXP_Image_Open](#)

[IFXP_Image_Get_Data](#)

[IFXP_Image_Close](#)

IFXP_Image_Close

This is called when Inflexion Runtime has finished with an image resource. The porting layer should release any resources associated with this session.

Usage

```
IFX_RETURN_STATUS IFXP_Image_Close(IFX_IMAGE_INFO* info);
```

Parameters

- **info**
Pointer to the IFX_IMAGE_INFO structure filled in during the call to IFXP_Image_Open.

Return Values

- **IFX_SUCCESS**
This method should always succeed.

Related Topics

[IFXP_Image_Open](#)

[IFXP_Image_Get_Data](#)

[IFXP_Image_Release_Data](#)

IFXP_Image_Check_Extension

This is called when the runtime is opening an image file with an extension that it does not recognize. The porting layer should check the extension to see if it is one supported by the custom image format unpacking set on this device.

Usage

```
IFX_RETURN_STATUS IFXP_Image_Check_Extension (const char *image_file);
```

Parameters

- **image_file**
Path of the image file to be opened, including the file extension.

Return Values

- **IFX_SUCCESS**
The file type is supported.
- **IFX_ERROR**
The file type is not supported.

Related Topics

[IFXP_Image_Open](#)

[IFXP_Image_Get_Data](#)

[IFXP_Image_Release_Data](#)

[IFXP_Image_Close](#)

Rendering

The following functions are used as part of the rendering pipeline (see the [Porting Overview](#) chapter for more details) to indicate where in the pipeline you are, and what actions to perform:

- [IFXP_Render_Start](#)
- [IFXP_Render_Background_Start](#)
- [IFXP_Render_Background](#)
- [IFXP_Render_Background_End](#)
- [IFXP_Render_Foreground_Start](#)
- [IFXP_Render_Foreground_End](#)
- [IFXP_Render_Display_Start](#)
- [IFXP_Render_Display](#)
- [IFXP_Render_Display_End](#)
- [IFXP_Render_End](#)

IFXP_Render_Start

This function is called at the start of the render cycle, allowing synchronization with an external system.

Usage

```
IFX_RETURN_STATUS IFXP_Render_Start (void);
```

Parameters

None.

Return Values

- IFX_SUCCESS
- IFX_ERROR

Any error - frame will be aborted.

Related Topics

[IFXP_Render_Background_Start](#)

[IFXP_Render_Foreground_Start](#)

[IFXP_Render_Display_Start](#)

[IFXP_Render_End](#)

IFXP_Render_Background_Start

This function is called before the background rendering step. For an asynchronous system, this function blocks until any active operations on the specified canvas have completed (for example, an asynchronous blit to display started on the previous cycle). If batch operations are supported, this function sets up the batch operation. For synchronous implementations, this always returns success immediately.

Usage

```
IFX_RETURN_STATUS IFXP_Render_Background_Start (IFXP_CANVAS* dst_canvas);
```

Parameters

- `dst_canvas`

This specifies the foreground canvas onto which the background is transferred.

Return Values

- `IFX_SUCCESS`
- `IFX_ERROR`

Any error.

Related Topics

[IFXP_Render_Start](#)

[IFXP_Render_Foreground_Start](#)

[IFXP_Render_Display_Start](#)

[IFXP_Render_End](#)

IFXP_Render_Background

This function copies portions of `src_canvas` (background) bounded by `src_rect` to `dst_canvas` (the canvas) at point (left,top). It is called zero or more times depending on the number of rectangles that require copying.

Usage

```
IFX_RETURN_STATUS IFXP_RenderBackground(  
    IFXP_CANVAS* dst_canvas,  
    IFX_INT32 left,  
    IFX_INT32 top,  
    IFXP_CANVAS* src_canvas,  
    IFXP_RECT* src_rect);
```

Parameters

- `dst_canvas`
This specifies the foreground canvas to either erase sections from (layered mode) or transfer sections of the background to.
- `top` and `left`
The top-left corner for the destination.
- `src_canvas`
This specifies the background canvas from which to copy
- `src_rect`
The source rectangle.

Return Values

- `IFX_SUCCESS`
- `IFX_ERROR`
Any error.

Description

For an asynchronous system, this function queues a blit command on each call, or where a queue is not available, this function ensures that appropriate action is taken to determine the correct execution of previous operations before starting a new operation.

Where multiple layers are used, this function erases the area on the foreground canvas to transparency instead of copying from the background to foreground.

For synchronous implementations, this function performs the requested blit and returns.



Note

This function is not always called, so the IFXP_Render_Background_Start and IFXP_Render_Background_End functions must take this into account and be able to continue in this case.

Related Topics

[IFXP_Render_Start](#)

[IFXP_Render_Foreground_Start](#)

[IFXP_Render_Background_Start](#)

[IFXP_Render_Background_End](#)

[IFXP_Render_Display_Start](#)

[IFXP_Render_End](#)

IFXP_Render_Background_End

This function is called after the background rendering step. For an asynchronous system, where batch operations are supported, this function sets the batch operation running. For synchronous implementations, this always returns success immediately.

Usage

```
IFX_RETURN_STATUS IFXP_Render_Background_End (IFXP_CANVAS* canvas);
```

Parameters

- `canvas`

This specifies the foreground canvas onto which the background is transferred.

Return Values

- `IFX_SUCCESS`
- `IFX_ERROR`

Any error.

Related Topics

[IFXP_Render_Start](#)

[IFXP_Render_Foreground_Start](#)

[IFXP_Render_Background_Start](#)

[IFXP_Render_Background](#)

[IFXP_Render_Display_Start](#)

[IFXP_Render_End](#)

IFXP_Render_Foreground_Start

This function is called before the internal foreground rendering step. For an asynchronous system, this function blocks until any active operations on the specified canvas have completed (for example, an asynchronous background blit). For synchronous implementations, this always returns success immediately. If the canvas requires locking, it must be done by this function.

Usage

```
IFX_RETURN_STATUS IFXP_Render_Foreground_Start (IFXP_CANVAS* canvas);
```

Parameters

- **canvas**
The canvas in use for this render cycle.

Return Values

- **IFX_SUCCESS**
- **IFX_ERROR**
Any error.

Related Topics

[IFXP_Render_Start](#)

[IFXP_Render_Foreground_End](#)

[IFXP_Render_Background_Start](#)

[IFXP_Render_Background](#)

[IFXP_Render_Display_Start](#)

[IFXP_Render_End](#)

IFXP_Render_Foreground_End

This function is called after the internal foreground rendering step. For synchronous implementations, this always returns success immediately. If the canvas requires unlocking, it is done by this function.

Usage

```
IFX_RETURN_STATUS IFXP_Render_Foreground_End (IFXP_CANVAS* canvas);
```

Parameters

- **canvas**
The canvas in use for this render cycle.

Return Values

- **IFX_SUCCESS**
- **IFX_ERROR**
Any error.

Related Topics

[IFXP_Render_Start](#)

[IFXP_Render_Foreground_Start](#)

[IFXP_Render_Background_Start](#)

[IFXP_Render_Background](#)

[IFXP_Render_Display_Start](#)

[IFXP_Render_End](#)

IFXP_Render_Display_Start

This function is called when the canvas is fully updated and ready for transfer to the display. There are no outstanding operations at this point due to these requirements, therefore, this function never blocks. Depending on the value of renderFull in the display structure, this function performs one of two operations:

- renderFull = 0

If batch operations are supported, this function sets up a batch operation.

- renderFull = 1

This function copies the whole canvas to display. Inflexion UI Runtime assumes the whole canvas has been transferred to the display by this call. No additional calls are made to IFXP_Render_Display or IFXP_Render_Display_End. Synchronous implementations returns success immediately.

Usage

```
IFX_RETURN_STATUS IFXP_Render_Display_Start (IFXP_CANVAS* canvas);
```

Parameters

- canvas

This specifies the foreground canvas that requires transferring to display.

Return Values

- IFX_SUCCESS
- IFX_ERROR

Any error.

Related Topics

[IFXP_Render_Start](#)

[IFXP_Render_Foreground_Start](#)

[IFXP_Render_Background_Start](#)

[IFXP_Render_Background](#)

[IFXP_Render_Display](#)

[IFXP_Render_End](#)

IFXP_Render_Display

This function copies the canvas data from `src_canvas`, bounded by `src_rRect` to the display starting at the left, top. If the display and canvas depth do not match, then color conversion is performed by this function. For an asynchronous system, this function can be called several times to queue multiple blits. Where a queue is not available, this function ensures that the correct execution of previous operations are performed before starting a new operation. For synchronous implementations, this function performs the requested blit and returns.

Usage

```
IFX_RETURN_STATUS IFXP_Render_Display (  
    IFX_INT32 left,  
    IFX_INT32 top,  
    IFXP_CANVAS* srcCanvas,  
    IFXP_RECT* srcRect);
```

Parameters

- `left`
- `top`
This specifies the top-left corner for where the source rectangle should be transferred onto the display. Note that this does not take into account any global offsets contained in the display structure.
- `src_canvas`
This specifies the foreground canvas that requires transferring to display.
- `src_rect`
This specifies the source rectangle that should be transferred from the foreground canvas.

Return Values

- `IFX_SUCCESS`
- `IFX_ERROR`
Any error.

Related Topics

[IFXP_Render_Start](#)[IFXP_Render_Foreground_Start](#)[IFXP_Render_Background_Start](#)[IFXP_Render_Background](#)[IFXP_Render_Display_Start](#)[IFXP_Render_End](#)

IFXP_Render_Display_End

This function is called at the end of a render cycle. For an asynchronous system, this starts any queued foreground operations. Where a queue is not available this function returns success immediately. On a synchronous system, this function returns success immediately. This function does not block waiting for asynchronous operations to complete, as this is done in IFXP_Render_Background_Start.

Usage

```
IFX_RETURN_STATUS IFXP_Render_Display_End (IFXP_CANVAS* canvas);
```

Parameters

- **canvas**
This specifies the foreground canvas that was transferred to display.

Return Values

- **IFX_SUCCESS**
- **IFX_ERROR**
Any error.

Related Topics

[IFXP_Render_Start](#)

[IFXP_Render_Display](#)

[IFXP_Render_Display_Start](#)

[IFXP_Render_End](#)

IFXP_Render_End

This function is called at the end of the render cycle, allowing synchronization with an external system.

Usage

```
IFX_RETURN_STATUS IFXP_Render_End (void);
```

Parameters

None.

Return Values

- IFX_SUCCESS
- IFX_ERROR

Any error - frame will be aborted.

Related Topics

[IFXP_Render_Start](#)

[IFXP_Render_Display_Start](#)

[IFXP_Render_Display_End](#)

Native Fonts

The following functions are used to control native fonts and create a bitmap canvas for a particular font and string:

- [IFXP_Text_Open_Font](#)
- [IFXP_Text_Close_Font](#)
- [IFXP_Text_Get_Width](#)
- [IFXP_Text_Get_Char_Height](#)
- [IFXP_Text_Canvas_Create](#)
- [IFXP_Text_Canvas_Destroy](#)

IFXP_Text_Open_Font

This function opens and initializes the named font with the face specified. If the specified font is not available, a default may be chosen.

Usage

```
IFX_RETURN_STATUS IFXP_Text_Open_Font (
    IFXP_FONT* font,
    IFXP_FONT_FACE face,
    IFX_WCHAR* font_name);
```

Parameters

- **font**
Pointer to the handle populated with the opened font.
- **face**
The typeface of the font. This should be one of:
 - IFXP_FONT_FACE_NORMAL - normal
 - IFXP_FONT_FACE_BOLD - bold
 - IFXP_FONT_FACE_ITALIC - italic
 - IFXP_FONT_FACE_BOLDITALIC bold and italicThis should be converted to a type corresponding to the appropriate face in the specific implementation.
- **font_name**
This specifies the requested font name. If this is not available, a default may be chosen.

Return Values

- **IFX_SUCCESS**
- **IFX_ERROR**
Any error.

Description

The IFXP_FONT_FACE is defined as follows:

```
typedef enum
{
    IFXP_FONT_FACE_NORMAL = 0,
    IFXP_FONT_FACE_BOLD,
    IFXP_FONT_FACE_ITALIC,
    IFXP_FONT_FACE_BOLDITALIC
} IFXP_FONT_FACE;
```

The IFXP_FONT type is a void*, so it is implementation specific.

Related Topics

[IFXP_Text_Close_Font](#)

IFXP_Text_Close_Font

This function closes a previously opened font.

Usage

```
IFX_RETURN_STATUS IFXP_Text_Close_Font (IFXP_FONT font);
```

Parameters

- font
Font handle created with IFXP_Text_Open_Font.

Return Values

- IFX_SUCCESS
- IFX_ERROR
Any error.

Related Topics

[IFXP_Text_Open_Font](#)

IFXP_Text_Get_Width

This function returns the width of the supplied string in pixels. The length parameter is provided so that the width can be calculated on part of the string (for example, when calculating caret position).

Usage

```
IFX_RETURN_STATUS IFXP_Text_Get_Width (  
    IFXP_FONT font,  
    IFX_WCHAR* text,  
    IFX_UINT32 in_length,  
    IFX_UINT32* rtlOut);
```

Parameters

- font
Font handle created with IFXP_Text_Open_Font.
- text
The text on which to perform the calculation.
- in_length
The number of characters in the “text” string to use in the width calculation. If this is “0”, then the whole string is used. When this is less than the total length of the text, the calculation applies to the first <in_length> characters in the text string.
- rtlOut
Pointer to the variable defining whether the text is principally rendered in a “right-to-left” direction, which should be set to 1 if true. Set to 0 if the text is “left-to-right”.

Return Values

- IFX_SUCCESS
- IFX_ERROR
Any error.

Related Topics

[IFXP_Text_Get_Char_Height](#)

IFXP_Text_Get_Char_Height

This function allows negotiation between the Inflexion UI Runtime and the porting layer regarding the height of the glyphs to be used in rendering text. The Runtime populates the “height” parameter with the required font height, and the porting layer must find a point size from the available set that is the closest match for that size.

If the precise font height can’t be matched, the porting layer should update the value of “height” with the value that is going to be used.

Usage

```
IFX_RETURN_STATUS IFXP_Text_Get_Char_Height (
    IFXP_FONT font,
    IFX_UINT32* height);
```

Parameters

- **font**
Font handle created with IFXP_TextOpenFont.
- **height**
Pointer to the variable populated with the character height.

Return Values

- **IFX_SUCCESS**
- **IFX_ERROR**
Any error.

Related Topics

[IFXP_Text_Get_Width](#)

IFXP_Text_Canvas_Create

This function creates an 8-bit gray scale buffer and renders the specified text into it using the font structure provided. It creates a new structure of type IFXP_TEXT and returns a pointer to it. If the text string should be rendered in a right-to-left direction (for example, if the **text** parameter contains Arabic unicode characters), the **rtl** parameter should be set, and the glyphs rendered to the canvas in the appropriate manner.

Usage

```
IFX_RETURN_STATUS IFXP_Text_Canvas_Create (
    IFXP_TEXT** textCanvas,
    IFXP_FONT font,
    IFX_WCHAR* text,
    IFX_INT32* rtlOut);
```

Parameters

- **text_canvas**
Points to the pointer to a text canvas structure.
- **font**
Font handle created with IFXP_TextOpenFont.
- **text**
The text to be rendered.
- **rtlOut**
Pointer to the variable defining whether the text is principally rendered in a “right-to-left” direction, which should be set to 1 if true. Set to 0 if the text is “left-to-right”.

Return Values

- IFX_SUCCESS
- IFX_ERROR
Any error.

Description

Where the native text render does not support 8 bpp gray scale the IFXP implementation renders internally, converting to 8 bpp gray scale. The IFXP_TEXT structure is defined as follows:

```
typedef struct ifxp_text
{
    void *alphaBuffer;
    IFX_UINT32 width;
    IFX_UINT32 height;
    void *internal;
} IFXP_TEXT;
```


- **alphaBuffer**
Populated with a pointer to an 8 bit per pixel single channel buffer. This buffer contains the text described by <text> as an alpha channel.
- **width and height**
Populated with the dimensions of the bitmap
- **internal**
Is not used by Inflexion UI and is provided for implementation-specific data.

Related Topics

[IFXP_Text_Canvas_Destroy](#)

IFXP_Text_Canvas_Destroy

This function destroys a previously created text canvas.

Usage

```
IFX_RETURN_STATUS IFXP_Text_Canvas_Destroy (IFXP_TEXT *canvas);
```

Parameters

- canvas
Text canvas created with IFXP_TextCanvasCreate.

Return Values

- IFX_SUCCESS
- IFX_ERROR
Any error.

Related Topics

[IFXP_Text_Canvas_Create](#)

Error Notes

The following function displays error notes when a critical error occurs:

- [IFXP_Display_Error_Note](#)

IFXP_Display_Error_Note

This function is called when a critical error occurs within Inflexion UI Runtime that must be relayed to the user. An example of this is when there is insufficient memory to perform an operation. The implementation of this function must pre-allocate any resources required to display the note because it is possible that no free memory remains on the system when called.

Usage

```
IFX_RETURN_STATUS IFXP_Display_Error_Note (IFX_WCHAR* error_text);
```

Parameters

- **error_text**
The text displayed in the error note. The maximum length of this string is 256 characters.

Return Values

- **IFX_SUCCESS**
If the error note is displayed.
- **IFX_ERROR**
The error note is not displayed.

Description

The function's precise behavior is left to your discretion. Once this function returns, Inflexion UI Runtime attempts to recover from the error and carry on (potentially from the “root” of the menu system), so the error note must behave in a modal manner as Inflexion UI wants to draw to the screen as soon as this function returns.

Related Topics

[IFXP_Error_Print](#)

FILE Functionality

The following functions are only required when platform files are enabled:

- [IFXP_File_Open](#)
- [IFXP_File_Read](#)
- [IFXP_File_Seek](#)
- [IFXP_File_Size](#)
- [IFXP_File_Close](#)
- [IFXP_File_Find_First](#)
- [IFXP_File_Find_Next](#)
- [IFXP_File_Find_Close](#)
- [IFXP_File_Create](#)
- [IFXP_File_Write](#)
- [IFXP_Dir_Create](#)

IFXP_File_Open

This function opens a file with the specified name. This function must perform conversion of the Inflexion UI Runtime drive letter notation into the platform's drive and path notation. For example, if Inflexion UI Runtime requests *C:\\Packages\\package.xml*, this function must convert that into the platform-specific location *C:\\dir_on_usb_stick\\Packages\\package.xml*.

Usage

```
IFX_RETURN_STATUS IFXP_File_Open (IFXP_FILE* handle, const char *file_name);
```

Parameters

- **handle**
Pointer to file handle to be populated.
- **file_name**
The path and filename to be opened in Inflexion UI Runtime notation.

Return Values

- **IFX_SUCCESS**
- **IFX_ERROR**
Any error.

Related Topics

[IFXP_File_Read](#)

[IFXP_File_Size](#)

[IFXP_File_Seek](#)

[IFXP_File_Close](#)

IFXP_File_Read

This function reads the size (in bytes) from the file described by the handle into the buffer at the addr address.

Usage

```
IFX_RETURN_STATUS IFXP_File_Read (  
    IFXP_FILE handle,  
    void* addr,  
    IFX_UINT32 size,  
    IFX_UINT32* bytes_read);
```

Parameters

- **handle**
The handle of a file previously opened with IFXP_File_Open.
- **addr**
The address at which data should be written.
- **size**
The number of bytes to read.
- **bytes_read**
Pointer to the variable to receive the number of bytes read.

Return Values

- **IFX_SUCCESS**
- **IFX_ERROR**
Any error.

Related Topics

[IFXP_File_Open](#)[IFXP_File_Seek](#)[IFXP_File_Size](#)[IFXP_File_Close](#)

IFXP_File_Seek

This function moves the file read pointer offset bytes from the location specified by seekMode. IFX_SEEK is defined in *nucleus/os/include/ui/ifxui_defs.h* as:

```
typedef enum ifx_seek
{
    IFX_SEEK_SET,
    IFX_SEEK_CUR,
    IFX_SEEK_END
} IFX_SEEK;
```

Usage

```
IFX_RETURN_STATUS IFXP_File_Seek (
    IFXP_FILE handle,
    IFX_INT32 offset,
    IFX_SEEK seek_mode);
```

Parameters

- **handle**
The handle of a file previously opened with IFXP_File_Open.
- **offset**
The number of bytes by which to seek from the location specified by seekMode.
- **seek_mode**
The location to seek from:
 - IFX_SEEK_SET seeks from the beginning of the file.
 - IFX_SEEK_CUR seeks from the current location.
 - IFX_SEEK_END seeks from the end of the file.

Return Values

- **IFX_SUCCESS**
Seeking was successful.
- **IFX_ERROR**
Seeking failed.

Related Topics

[IFXP_File_Read](#)

[IFXP_File_Size](#)

[IFXP_File_Close](#)

IFXP_File_Size

This function returns the size in bytes of a file described by the handle.

Usage

```
IFX_RETURN_STATUS IFXP_File_Size (  
    IFXP_FILE handle,  
    IFX_UINT32* file_size);
```

Parameters

- **handle**
The handle of a file previously opened with `IFXP_File_Open`.
- **file_size**
Pointer to the variable to receive the file size.

Return Values

- `IFX_SUCCESS`
- `IFX_ERROR`
Any error.

Related Topics

[IFXP_File_Open](#)

[IFXP_File_Close](#)

[IFXP_File_Seek](#)

IFXP_File_Close

This function closes a file described by the handle and frees any resources with which it is associated.

Usage

```
IFX_RETURN_STATUS IFXP_File_Close (IFXP_FILE handle);
```

Parameters

- **handle**
The handle of a file previously opened with `IFXP_File_Open`.

Return Values

- **IFX_SUCCESS**
This function must always return success (even if the file could not be closed for some reason).

Related Topics

[IFXP_File_Open](#)

IFXP_File_Find_First

This function initializes the find structure at info_ptr and searches for the first occurrence of search_text. As with the IFXP_File_Open function, this function must convert the Inflexion UI drive notation into the platform specific notation. All instances of searchText begin with an absolute path of the form <drive letter>:\.

Usage

```
IFX_RETURN_STATUS IFXP_File_Find_First (
    IFXP_FILE* handle,
    IFXP_SEARCH* infoPtr,
    char* search_text);
```

Parameters

- handle
A pointer to a file handle populated by this function.
- info_ptr
A pointer to an IFXP_SEARCH structure. This is owned by the caller.
- search_text
The text string for which to search.

Return Values

- IFX_SUCCESS
The searchText is matched.
- IFX_ERROR
The searchText is not found.

Description

IFXP_SEARCH is defined as follows:

```
typedef struct _ifxp_search
{
    IFXP_INFO info;
    char fileName[IFXP_MAX_FILE_NAME_LEN];
    void* internal;
} IFXP_SEARCH;
```

IFXP_INFO is defined as follows:

```
typedef enum _ifxp_info
{
    IFXP_INFO_INVALID,
    IFXP_INFO_DIRECTORY,
    IFXP_INFO_FILE
} IFXP_INFO;
```

- **info**
Used to inform whether a search match is a file or directory.
- **fileName**
The name of the file or directory matched. This is the fileName only. It does not include the path. For example: search *Text* = *C:\\Packages\\main_demo_top_** and the directory, *main_demo_top_theme1* exists, fileName is populated with *main_demo_top_theme1*.
- **internal**
Available for use by the Porting Layer for any data associated with this search instance. Any allocated data should be allocated by this function, and freed in IFXP_File_Find_Close.

Related Topics

[IFXP_File_Seek](#)

[IFXP_File_Find_Next](#)

[IFXP_File_Find_Close](#)

IFXP_File_Find_Next

This function searches for the next occurrence of the search term initialized in IFXP_File_Find_First.

Usage

```
IFX_RETURN_STATUS FXP_File_Find_Next (
    IFXP_FILE handle,
    IFXP_SEARCH* info_ptr);
```

Parameters

- **handle**
A search file handle previously initialized with IFXP_File_Find_First.
- **info_ptr**
A pointer to an IFXP_SEARCH structure.

Return Values

- **IFX_SUCCESS**
A match is found.
- **IFX_ERROR**
No match found.

Related Topics

[IFXP_File_Seek](#)

[IFXP_File_Find_First](#)

[IFXP_File_Find_Close](#)

IFXP_File_Find_Close

This function closes a file search handle and frees all data with which it is associated.

Usage

```
IFX_RETURN_STATUS IFXP_File_Find_Close (  
    IFXP_FILE handle,  
    IFXP_SEARCH* info_ptr );
```

Parameters

- **handle**
A search file handle previously initialized with `IFXP_File_Find_First`.
- **info_ptr**
A pointer to an `IFXP_SEARCH` structure.

Return Values

- **IFX_SUCCESS**
This function always succeeds.

Related Topics

[IFXP_File_Seek](#)

[IFXP_File_Find_First](#)

[IFXP_File_Find_Next](#)

IFXP_File_Create

This function should create a file on the platform filesystem with the specified name. Currently only used when the Runtime is built in OpenGL 2.0 rendering mode, and where the GPU driver supports the glGetProgramBinaryOES extension.

Usage

```
IFX_RETURN_STATUS   IFXP_File_Create (IFXP_FILE *handle, const char
                                     *file_name);
```

Parameters

- **handle**
The handle of the newly-created file.
- **file_name**
Name and location of the file to create.

Return Values

- **IFX_SUCCESS**
The file is created, and 'handle' contains a valid file handle.
- **IFX_ERROR**
The file could not be created.

Related Topics

[IFXP_File_Write](#)

IFXP_File_Write

This function should append the data in the buffer passed in to the file referenced by the specified handle. Currently only used when the Runtime is built in OpenGL 2.0 rendering mode, and where the GPU driver supports the glGetProgramBinaryOES extension.

Usage

```
IFX_RETURN_STATUS IFXP_File_Write (IFXP_FILE handle, void *addr, IFX_UINT32  
    size, IFX_UINT32 *bytes_written);
```

Parameters

- **handle**
The handle of the file created using IFXP_File_Create.
- **addr**
The location of the buffer containing the data to write.
- **size**
The number of bytes to write.
- **bytes_written**
Out parameter - this contains the number of bytes actually written to file.

Return Values

- **IFX_SUCCESS**
File write operation succeeded.
- **IFX_ERROR**
Any error.

Related Topics

[IFXP_File_Create](#)

IFXP_Dir_Create

This function should create a directory on the platform filesystem with the specified name. Currently only used when the Runtime is built in OpenGL 2.0 rendering mode, and where the GPU driver supports the glGetProgramBinaryOES extension.

Usage

```
IFX_RETURN_STATUS IFXP_Dir_Create (const char *new_dir);
```

Parameters

- `new_dir`
Name and location of the directory to create.

Return Values

- `IFX_SUCCESS`
The directory is created.
- `IFX_ERROR`
The directory could not be created.

Related Topics

[IFXP_File_Create](#)

Timer Management

The following functions are used to initialize, shutdown and schedule the timer used by Inflexion UI Runtime. Inflexion UI Runtime does not control the starting and stopping of the timer - this is an implementation detail left to the porting layer.

- [IFXP_Timer_Schedule](#)
- [IFXP_Timer_Get_Current_Time](#)

IFXP_Timer_Schedule

This function schedules the specified timer to expire at “time” ms as an absolute system time at the latest. If the timer is already scheduled to expire before this time, the timer will not be modified. The time reference should be the system timer in milliseconds.

Usage

```
IFX_RETURN_STATUS IFXP_Timer_Schedule (  
    IFX_UINT32 time_upper,  
    IFX_UINT32 time_lower);
```

Parameters

- **time_upper**
The upper DWORD of the 64-bit absolute time in ms that the timer will expire (at the latest).
- **time_lower**
The lower DWORD of the 64-bit absolute time in ms that the timer will expire (at the latest).

Return Values

- **IFX_SUCCESS**
- **IFX_ERROR**
Any error.

Related Topics

[IFXP_Timer_Get_Current_Time](#)

IFXP_Timer_Get_Current_Time

This function populates the variable time with the current system time in milliseconds.

Usage

```
IFX_RETURN_STATUS   IFXP_Timer_Get_Current_Time (IFX_UINT32 *time_upper,  
                                                IFX_UINT32 *time_lower);
```

Parameters

- **time_upper**
Pointer to a variable to hold the upper DWORD of the 64-bit system time in milliseconds
- **time_lower**
Pointer to a variable to hold the lower DWORD of the 64-bit system time in milliseconds

Return Values

- **IFX_SUCCESS**
- **IFX_ERROR**
Any error

Related Topics

[IFXP_Timer_Schedule](#)

Benchmarking

The following functions are used to manage benchmarking events sent out from Inflexion UI Runtime:

- [IFXP_Benchmarking_Signal](#)

IFXP_Benchmarking_Signal

This function is provided for benchmarking Inflexion UI Runtime. The events are defined in *nucleus/os/ui/inflexionui/engine/ifxui_porting.h* and represent particular events during Inflexion UI Runtime's operation. Use of the information provided is optional.

This function will only be linked to and called when either `IFX_ENABLE_BENCHMARKING_HEAP_USAGE` or `IFX_ENABLE_BENCHMARKING_FRAME_RATE` is defined.

Usage

```
IFX_RETURN_STATUS IFXP_Benchmarking_Signal (IFX_UINT32 signal);
```

Parameters

- `signal`
Type of event that has occurred.

Return Values

- `IFX_SUCCESS`
- `IFX_ERROR`
Any error.

Debug Support

The following function can be used to output debug messages from Inflexion UI Runtime to a debug channel:

- [IFXP_Error_Print](#)

IFXP_Error_Print

This function allows error messages to be propagated onto a debug channel, for example serial output. This function never returns failure.

Usage

```
IFX_RETURN_STATUS IFXP_Error_Print ( const IFX_WCHAR *errorString );
```

Parameters

- `errorString`
Error string describing the error in engineering English.

Return Values

- `IFX_SUCCESS`

Related Topics

[IFXP_Display_Error_Note](#)

Memory Management

The following functions manage memory used by Inflexion UI Runtime. Although a mechanism is provided to separate Inflexion UI Runtime, String and General memory pools; it depends on the implementation whether this mechanism is used.

- [IFXP_Mem_Allocate](#)
- [IFXP_Mem_Deallocate](#)

IFXP_Mem_Allocate

This function allocates the requested amount of memory from the specified memory pool. If the size is zero, or the pointer is NULL then IFX_ERROR is returned.

Ensure that allocations are 32-bit aligned.

Usage

```
IFX_RETURN_STATUS IFXP_Mem_Allocate (  
    IFXP_MEM_USE usage,  
    IFX_UINT32 size,  
    void **ptr);
```

Parameters

- **usage**
Specifies what Inflexion UI Runtime will use the memory for to allow the segmentation of memory for efficiency and or protection.
- **size**
The number of bytes to allocate.
- **ptr**
Pointer to a pointer to store the start of the allocated block.

Return Values

- **IFX_SUCCESS**
- **IFX_ERROR**
Any error.

Description

The IFXP_MEM_USE type used by this function is defined as follows:

```
typedef enum{  
    IFXP_MEM_ENGINE,  
    IFXP_MEM_EXTERNAL,  
    IFXP_MEM_STRING  
} IFXP_MEM_USE;
```

- **IFXP_MEM_ENGINE**
Is used for internal Inflexion UI engine allocations, excluding string allocations. It specifies that the Inflexion UI engine-owned memory pool will be used.
- **IFXP_MEM_EXTERNAL**
Is used for external allocations, for example from integrated modules. It specifies that the general system memory pool will be used.

- **IFXP_MEM_STRING**

Is used for allocating internal strings. It specifies that the string memory pool will be used.

Related Topics

[IFXP_Mem_Deallocate](#)

IFXP_Mem_Deallocate

This function frees the memory pointed to by ptr.

Usage

```
IFX_RETURN_STATUS IFXP_Mem_Deallocate (IFXP_MEM_USE usage, void *ptr);
```

Parameters

- **usage**
Specifies the usage type requested at allocation.
- **ptr**
The pointer to the memory block to be freed.

Return Values

- **IFX_SUCCESS**

Related Topics

[IFXP_Mem_Allocate](#)

Mutexes

The following functions are used to configure and to perform lock/unlock operations on the Mutex:

- [IFXP_Mutex_Create](#)
- [IFXP_Mutex_Destroy](#)
- [IFXP_Mutex_Lock](#)
- [IFXP_Mutex_Unlock](#)

IFXP_Mutex_Create

This function is responsible for creating a Mutex. This function allocates any required resources. The mutex variable is populated with a handle that Inflexion UI Runtime uses to identify the mutex in future calls.

The IFXP_MUTEX type used by this function is defined as follows:

```
typedef void* IFXP_MUTEX;
```

Usage

```
IFX_RETURN_STATUS IFXP_Mutex_Create(  
    IFXP_MUTEX* mutex,  
    const IFX_WCHAR *name);
```

Parameters

- **mutex**
Pointer to MUTEX internal object. This pointer is used to keep track of the Mutex. This pointer is implementation specific.
- **name**
Name of the mutex to create. This is guaranteed to be a unique string of maximum 8 characters (including the NULL terminator).

Return Values

- **IFX_SUCCESS**
- **IFX_ERROR**
Any error.

Related Topics

[IFXP_Mutex_Destroy](#)

[IFXP_Mutex_Lock](#)

[IFXP_Mutex_Unlock](#)

IFXP_Mutex_Destroy

This function destroys the Mutex. This function deallocates any resources allocated during creation.

Usage

```
IFX_RETURN_STATUS IFXP_Mutex_Destroy(IFXP_MUTEX mutex);
```

Parameters

- **mutex**
Pointer to MUTEX internal object. This pointer is used to keep track of the Mutex. This pointer is implementation specific.

Return Values

- **IFX_SUCCESS**

Related Topics

[IFXP_Mutex_Create](#)

[IFXP_Mutex_Lock](#)

[IFXP_Mutex_Unlock](#)

IFXP_Mutex_Lock

This function is called by Inflexion UI Runtime to acquire a lock on the Mutex for the current executing Thread/Task.

Implementation of this function is specific to the platform. The timeout is provided in ms; however there is a special case where an infinite wait is required.

Usage

```
IFX_RETURN_STATUS IFXP_Mutex_Lock(  
    IFXP_MUTEX mutex,  
    IFX_UINT32 timeout);
```

Parameters

- **mutex**
Pointer to MUTEX internal object. This pointer is used to keep track of the Mutex. Implementation of this pointer is platform specific.
- **timeout**
Maximum time in milliseconds to wait for the availability of the Mutex.

Where an infinite wait is required, Inflexion UI Runtime will pass IFXP_MUTEX_INFINITE as the timeout. The implementation treats this value as infinite.

Where zero is specified, the function checks the mutex and returns immediately if a lock is not possible.

Return Values

- **IFX_SUCCESS**
- **IFXP_MUTEX_TIMEOUT**
The mutex request timed out.
- **IFX_ERROR**
Any error.

Related Topics

[IFXP_Mutex_Create](#)

[IFXP_Mutex_Destroy](#)

[IFXP_Mutex_Unlock](#)

IFXP_Mutex_Unlock

This function releases an acquired Mutex attached with the current executing thread/task. The Mutex that is to be released is identified using the supplied parameter.

Usage

```
IFX_RETURN_STATUS IFXP_Mutex_Unlock(IFXP_MUTEX mutex);
```

Parameters

- **mutex**
Pointer to MUTEX internal object. This pointer is used to keep track of the Mutex. Implementation of this pointer is platform specific.

Return Values

- **IFX_SUCCESS**
- **IFX_ERROR**
Any error.

Related Topics

[IFXP_Mutex_Create](#)

[IFXP_Mutex_Destroy](#)

[IFXP_Mutex_Lock](#)

Other Methods

The following methods complete the porting layer API:

- [IFXP_Check_GL_Extension](#)

IFXP_Check_GL_Extension

This function is called by Inflexion UI Runtime to query the set of Open GL extensions supported by the device. If the device doesn't support a particular extension, or if that feature shouldn't be offered to the runtime, this method should return IFX_ERROR. If the extension is supported, the method should return IFX_SUCCESS.

Usage

```
IFX_RETURN_STATUS IFXP_Mutex_Lock(const char* ext_name);
```

Parameters

- **ext_name**
Null terminated string indicating the extension to check (for example "GL_OES_get_program_binary").

Return Values

- **IFX_SUCCESS**
Extension supported.
- **IFX_ERROR**
Extension not supported.

Validating a New Porting Layer

If you have re-implemented the porting layer or any portion of it, it is strongly recommended that you thoroughly test the new implementation to maintain the stability of Inflexion UI Runtime. This section contains some notes and suggestions to use when you construct the test plan. The initialization, canvas management, and shutdown of the porting layer are tested implicitly when the Inflexion UI Runtime is started and stopped.

The following additional tests should be performed as a general check:

- Enable `IFX_USE_PLATFORM_FILES` and verify that a base package loads and runs from the file system.
- Enable `IFX_USE_NATIVE_FONTS` and verify the following:
 - One character length strings
 - Long strings
 - Unicode (non-ASCII) strings
 - Rotation and Scaling of a text element
- Check that the current display mode can be changed via `IFXP_Set_Configuration` if the platform supports multiple display modes.
- Check the displayed image for red and blue color reversal.
- Reduce the amount of memory available to the Inflexion UI Runtime to test the error note functionality.
- Where OpenGL is supported, compile and run in OpenGL direct mode.

Chapter 11

Included Third-party Software

In this chapter:

FreeType	301
Harfbuzz	304
NUbidi	305
UCDN (Unicode Database and Normalization)	305

FreeType

FreeType (<http://www.freetype.org>) is an open source, public domain implementation of FreeType font engine. FreeType 2 is written in industry-standard ANSI C and is compatible with any compliant C compiler. Apart from standard ANSI C library, FreeType doesn't have any external dependencies and can be easily integrated with any kind of system. It is designed to be small, efficient, highly customizable, and portable while capable of producing high-quality output (glyph images). Due to its compact size and modularized implementation, it's very suitable for the embedded applications.

FreeType Features

The following features are provided by FreeType:

- FreeType 2 provides a simple and easy-to-use API to access font content in a uniform way, independently of the file format. Additionally, some format-specific APIs can be used to access special data in the font file.
- FreeType 2 supports scalable font formats like TrueType or Type 1 natively and can return the outline data.
- The design of FreeType 2 is based on modules that can be either linked statically to the library at compile time, or loaded on demand at runtime.
- FreeType 2 doesn't use static writable data (for example, it can be run from ROM directly) and is suitable for embedded platforms.
- Size of the FreeType library can be reduced by selecting only the required modules.
- FreeType 2 is capable of producing a high-quality monochrome bitmap, or anti-aliased pixmap, using 256 levels of "gray".

- FreeType 2 supports all the character mappings defined by the TrueType and OpenType specification.
- FreeType 2 provides its own caching subsystem for latest versions. It can be used to cache either face instances or glyph images efficiently.
- By default, FreeType 2 supports the following font formats:

TrueType fonts (and collections)

- Type 1 fonts
- CID-keyed Type 1 fonts
- CFF fonts
- OpenType fonts (both TrueType and CFF variants)
- SFNT-based bitmap fonts
- X11 PCF fonts
- Windows FNT fonts
- BDF fonts (including anti-aliased ones)
- PFR fonts
- Type 42 fonts (limited support)

Design

The following are the main design features of FreeType. These features are also listed at <http://freetype.sourceforge.net/freetype2/docs/design/design-1.html>:

- High portability. The library is designed to be able to run on any kind of environment.
- Extensibility. New features can be added with the least modifications in the FreeType library's code base. This results in extremely simple design where nearly all operations are provided by modules.
- Customization. It is easy to build a version of the library that only contains the features needed by a specific project. It's really important when integrated with embedded applications.
- Compactness and efficiency. FreeType targets embedded systems with low processor and memory resources. FreeType library was designed in a form to run under low processor and memory resources.

Modes of Operation

In order to produce Glyph images, FreeType requires font files, which can be supplied in the following three ways:

- Reading font files directly from Font files stored on disk using open source FreeType
- Reading font files from memory buffer
- Reading font files from font files stored on disk and loading them to memory buffer before passing to FreeType

To enable FreeType read font files directly from disk, you need to place the font files on disk and pass the appropriate path during FreeType font face initialization. The library API `FT_New_Face` accepts the file name and path to read font files. In this mode, FreeType will access the file system to read the font file whenever there is request for Glyphs. This mode is useful when a font file is very large; however access is slower due to file system overheads.

To use fonts from a memory buffer, you need to convert the font file to a header file containing binary font data. Then buffer is passed to FreeType at initialization. FreeType APIs like `FT_New_Memory_Face` open the font using the font buffer provided as a parameter. FreeType then processes characters based on the given font. This mode is the default mode of operation.

To improve open source FreeType performance when reading fonts from the files on storage media, the Inflexion UI Runtime component provides an optimized way by first loading the complete font file into memory buffer and then pass it to FreeType library using the normal `FT_New_Memory_Face` API. In order to read font files directly from disk and load to memory buffer, you must enable the `use_platform_files` option in a ReadyStart configuration file.

Adding New Font File Support

To add support for new font files (reading from the filesystem):

- Under this mode, the font name is directly mapped to the file on filesystem by concatenating the driver path and the file extension. In order to change the default drive path, the define `DEFAULT_FILENAME_PREFIX` and to change the default file extension, the define `DEFAULT_FILENAME_EXTENSION` in file `nucleus/os/ui/inflexionui/porting/src/ixui_porting_font.c` should be modified accordingly.

To add support of a new font using the binary memory buffer:

- Convert the TTF file (or any font file extension) to a binary header file containing array of font data.
- Include this file in `nucleus/os/ui/inflexionui/porting/src/ixui_porting_font.c`.

- Modify the `IFXP_Text_Open_Font` in file *nucleus/os/ui/inflexionui/porting/src/ifxui_porting_font.c* adding an appropriate `if` block to map the font name to the new font header array.
- Alternatively, the macro `DEFAULT_FONT_HEADER_FILE_NAME` in file *nucleus/os/ui/inflexionui/porting/src/ifxui_porting_font.c* can be modified to change the default file name to the one you provided, although this will override the default font.

To add support for new font files (reading from the filesystem and loading to memory buffer):

- Modify `PL_Get_Font_File_Path` in file *nucleus/os/ui/inflexionui/porting/src/ifxui_porting_font.c* to add an “`if`” block for the given font. This function maps the font name used in theme files to the actual filename and extension of a TTF file (or any font file extension).
- Alternatively, the macro `DEFAULT_FONT_FILE_NAME` in file *nucleus/os/ui/inflexionui/porting/src/ifxui_porting_font.c* can be modified to change the default file name to the one you provided, although this overrides the default font.

Configuring FreeType Caching

Open source FreeType provides caching feature to improve performance. This includes caching of glyphs, glyphs index, and so forth. Refer to the default demonstration files to see the implementation. Refer to the open source FreeType documentation for more information on the caching feature.

By default, the Inflexion UI Runtime component uses the caching feature of open source FreeType. It is recommended that you configure the size of the cache when using fonts other than the default one. To disable caching support, set the `USE_FT_CACHE_SUPPORT` macro in the *nucleus/os/ui/inflexionui/porting/src/ifxui_porting_font.c* file to `IFX_FALSE`.

Function Reference

All documentation for the open source FreeType project is available at <http://freetype.sourceforge.net/freetype2/documentation.html>

Harfbuzz

Harfbuzz is an OpenType shaping engine that uses information in the specified font along with information from the Unicode database to select the correct glyph based on the text string context. This includes use of ligatures, alternate forms of characters, and reversed ordering in the case of “right-to-left” languages, and is the basis of the support in Inflexion of Arabic, Urdu, Hebrew, and similar languages.

Harfbuzz is a third-party open-source library. For more details please visit:

<http://www.freedesktop.org/wiki/Software/HarfBuzz>

Usage Notes

Harfbuzz is only used within the Native Fonts porting layer, and is used to translate the text string held by the runtime into the appropriate glyph sequence to be drawn using Freetype. If you are writing your own Native Font porting layer implementation you will need to process the text strings in a similar manner if you wish to support complex and/or “right-to-left” languages (such as Arabic). You will also need to inform the runtime if a string is read in a “right-to-left” manner via the `rtlOut` parameters in the `IFXP_Text_Get_Width` and `IFXP_Text_Canvas_Create` methods if the runtime is to support “marquee” scrolling of long text strings correctly.

Harfbuzz is built and statically linked as part of the runtime build process.

Version Shipped with Inflexion UI Runtime

Inflexion UI Runtime distributes version 0.9.0 of Harfbuzz ng, with minor modifications to suit use on multiple platforms (the modifications do not affect the operation of the library).

Harfbuzz Licensing

HarfBuzz is licensed under the so-called “Old MIT” license.

NUbidi

NUbidi is an implementation of the Unicode Bidirectional Algorithm, which is a method for breaking down a text string into substrings containing characters designed to be read in a common direction. This allows correct rendering of mixed left-to-right and right-to-left strings.

Usage Notes

NUbidi is only used within the Native Fonts porting layer, and is used to break the text string held by the runtime into directional “runs”, each of which can then be passed to Harfbuzz for shaping, and then to Freetype for rendering. If you are writing your own Native Font porting layer implementation you will need to process the text strings in a similar manner if you wish to support bi-directional strings in the runtime.

NUbidi Licensing

As the NUbidi library is written by Mentor Graphics, it is licensed under the same terms as the Inflexion UI Runtime.

UCDN (Unicode Database and Normalization)

UCDN is a Unicode support library. Currently, it provides access to basic character properties contained in the Unicode Character Database and low-level normalization functions.

Usage Notes

UCDN is only used within the Native Fonts porting layer, and is used to supply information to the Harfbuzz library. An alternative is to use the “ICU4C” library if that is already present on the target device, which may save some ROM footprint. To do this, you will have to change the definition of HB_SCRIPT_UNICODE, and change the second parameter to the call to `hb_buffer_set_unicode_funcs` in the file *ifxui_porting_file.c*.

Version Shipped with Inflexion UI Runtime

There is no release for the UCDN library. Inflexion UI Runtime distributes the UCDN files from the “master” branch of the github repository corresponding to the commit on August 19th, 2012, reference 92e82cddb6c11df5dc606388f314abec27c98e70.

UCDN Licensing

UCDN is licensed under the ISC license.

Third-Party Information

This document provides information on open source and third-party software that may be included in Mentor Embedded software products

- This software application may include Java 3D version 1.5.1 third-party software.

Java 3D version 1.5.1 is distributed under the terms of the Java 3D License version 1.5.1 and is distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the license for the specific language governing rights and limitations under the license. You can view a copy of the license at: `<install_root>/docs/legal/java3d_1.5.1.pdf`. Portions of this software may be subject to the Java Distribution License version 1.0. You can view a copy of the license at: `<install_root>/docs/legal/java_dist_1.0.pdf`. Portions of this software may be subject to the Wild Magic Software License version 2. You can view a copy of the license at: `<install_root>/docs/legal/wild_magic_2_r1.pdf`. Java 3D version 1.5.1 may be subject to the following copyrights:

© 1991, 1997 Digital Equipment Corporation, Maynard, Massachusetts.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software.

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL DIGITAL EQUIPMENT CORPORATION BE LIABLE FOR ANY CLAIM, DAMAGES, INCLUDING, BUT NOT LIMITED TO CONSEQUENTIAL OR INCIDENTAL DAMAGES, OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of Digital Equipment Corporation shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization from Digital Equipment Corporation.

© 2007 Sun Microsystems, Inc. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistribution of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistribution in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Sun Microsystems, Inc. or the names of contributors may be used to endorse or promote products derived from this software without specific prior written permission.

This software is provided “AS IS,” without a warranty of any kind. ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN MICROSYSTEMS, INC. (“SUN”) AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE THIS SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You acknowledge that this software is not designed, licensed or intended for use in the design, construction, operation or maintenance of any nuclear facility.

© 2007 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at <http://www.sun.com/patents> and one or more additional patents or pending patent applications in the U.S. and in other countries.

U.S. Government Rights - Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

Use is subject to license terms.

This distribution may include materials developed by third parties.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and in other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, Java, Solaris, Java 3D, the 100% Pure Java logo, the Duke logo and the Java Coffee Cup logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

This product is covered and controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries.

Nuclear, missile, chemical biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists is strictly prohibited.

© 2007 The Khronos Group Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and/or associated documentation files (the "Materials"), to deal in the Materials without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Materials, and to permit persons to whom the Materials are furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Materials.

THE MATERIALS ARE PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE MATERIALS OR THE USE OR OTHER DEALINGS IN THE MATERIALS.

© 1996 by Jef Poskanzer <jef@acme.com>. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT

SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- This software application may include KTable version 2.1.3 third-party software.

KTable version 2.1.3 is distributed under the terms of the Eclipse Public License version 1.0 and is distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. You can view a copy of the Eclipse Public License at: [<install_root>/docs/legal/eclipse_1.0.pdf](http://install_root/docs/legal/eclipse_1.0.pdf). To obtain a copy of the KTable source code, or modifications to KTable, if any, send a request to request_sourcecode@mentor.com. KTable version 2.1.3 may be subject to the following copyrights:

© 2004 by Friederich Kupzog Elektronik & Software

All rights reserved. This program and the accompanying materials are made available under the terms of the Eclipse Public License v1.0 which accompanies this distribution, and is available at <http://www.eclipse.org/legal/epl-v10.html>.

- This software application may include Neuquant version 1994 third-party software, which is distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. Neuquant version 1994 may be subject to the following copyrights:

© 1994 Anthony Dekker

NEUQUANT Neural-Net quantization algorithm by Anthony Dekker, 1994.

See “Kohonen neural networks for optimal colour quantization” in “Network: Computation in Neural Systems” Vol. 5 (1994) pp 351-367. for a discussion of the algorithm. See also <http://members.ozemail.com.au/~dekker/NEUQUANT.HTML> Any party obtaining a copy of these files from the author, directly or indirectly, is granted, free of charge, a full and unrestricted irrevocable, world-wide, paid up, royalty-free, nonexclusive right and license to deal in this software and documentation files (the “Software”), including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons who receive copies from any such party to do so, with the only requirement being that this copyright notice remain intact.

- This software application may include M+ Outline Fonts version 18 third-party software, which is distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied.
- FreeType version 2.3.8

This software application may include FreeType version 2.3.8 third-party software. FreeType v2.3.8 is distributed under the terms of the FreeType Project License and is distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the license for the specific language governing rights and limitations under the license. You can view a copy of the license at: [<install_root>/docs/legal/freetype_2006.pdf](http://install_root/docs/legal/freetype_2006.pdf). FreeType v2.3.8 may be subject to the following copyrights:

© 1990, 1994, 1998 The Open Group

Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation.

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE OPEN GROUP BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR

OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

© 2005, 2007, 2008 by George Williams

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- STLport version 5.1.4

This software application may include STLport version 5.1.4 third-party software, which is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. STLport version 5.1.4 may be subject to the following copyrights:

© 1994 Hewlett-Packard Company

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Hewlett-Packard Company makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

© 1996-1999 Silicon Graphics Computer Systems, Inc.

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Silicon Graphics makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

© 1997 Moscow Center for SPARC Technology

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Moscow Center for SPARC Technology makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

© 1997-1998 Mark of the Unicorn, Inc.

-Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Mark of the Unicorn, Inc. makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

© 2000 Jens Maurer

Permission to use, copy, modify, sell, and distribute this software is hereby granted without fee provided that the above copyright notice appears in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Jens Maurer makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

- This software application may include Android SDK fonts version 2.1 third-party software.
- Android SDK fonts version 2.1 is distributed under the terms of the Apache License version 2.0 and is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the license for the specific language governing rights and limitations under the license. You can view a copy of the license at: `<install_root>/docs/legal/apache_2.0.pdf`. This software application may include Android version 1.6 third-party software.

Android version 1.6 is distributed under the terms of the Apache License version 2.0 and is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the license for the specific language governing rights and limitations under the license. You can view a copy of the license at: `<install_root>/docs/legal/apache_2.0.pdf`.

- This software application may include FreeType version 2.3.8 third-party software.

FreeType version 2.3.8 is distributed under the terms of the FreeType Project License and is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the license for the specific language governing rights and limitations under the license. You can view a copy of the license at: `<install_root>/docs/legal/freetype_2006.pdf`. FreeType version 2.3.8 may be subject to the following copyrights:

© 2005, 2007, 2008 by George Williams

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- This software application may include Java Runtime Environment version 1.6.0_20 third-party software.

Java Runtime Environment version 1.6.0_20 is distributed under the terms of the Sun Microsystems, Inc. Binary Code License Agreement for the JAVA SE RUNTIME ENVIRONMENT (JRE) VERSION 6 and JAVAFX RUNTIME VERSION 1 and is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the license for the specific language governing rights and limitations under the license. You can view a copy of the license at: `<install_root>/docs/legal/sun_binary_JRE_6.pdf`. Additional copyright notices and license terms applicable to portions of Java Runtime Environment version 1.6.0_20 can be found here: `<install_root>/THIRDPARTYLICENSEREADME.txt`.

- This software application may include Eclipse SDK version 3.6 third-party software.

SDK version 3.6 is distributed under the terms of the Eclipse Public License version 1.0 and is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the about.html files for information

about third-party software, licensing, and copyright information. These can be displayed from the Help menu item and selecting “About Inflexion UI Express”. You can view a copy of the Eclipse Public License version 1.0 at: `<install_root>/docs/legal/eclipse_1.0.pdf`. Portions of this software may be subject to the GNU Lesser General Public License version 2.1. You can view a copy of the license at: `<install_root>/docs/legal/gnu_lgpl_2.1.pdf`. Mozilla and the Mozilla binding are distributed under the terms of the GNU Lesser General Public License. Portions of this software may be subject to the Apache License version 2.0. You can view a copy of the Apache License at `<install_root>/docs/legal/apache_2.0.pdf`. Portions of this software may be subject to the Apache License version 1.1. You can view a copy of the Apache License at `<install_root>/docs/legal/apache_1.1.pdf`. Portions of this software may be subject to the W3C Software Notice and License. You can view a copy of the W3C Software License at `<install_root>/docs/legal/w3c_2002.pdf`. To obtain a copy of the Eclipse Modeling Tool source code, send a request to request_sourcecode@mentor.com.

- This software application may include Eclipse Modeling Framework SDK version 2.6.0 third-party software.

Eclipse Modeling Framework SDK version 2.6.0 is distributed under the terms of the Eclipse Public License version 1.0 and is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the about.html files for information about third-party software, licensing, and copyright information. These can be displayed from the Help menu item and selecting “About Inflexion UI Express”. You can view a copy of the Eclipse Public License version 1.0 at: `<install_root>/docs/legal/eclipse_1.0.pdf`. Portions of this software may be subject to the Apache License version 2.0. You can view a copy of the Apache License at `<install_root>/docs/legal/apache_2.0.pdf`. Portions of this software may be subject to the Apache License version 1.1. You can view a copy of the Apache License at `<install_root>/docs/legal/apache_1.1.pdf`. To obtain a copy of the Eclipse Modeling Tool source code, send a request to request_sourcecode@mentor.com.

- This software application may include EMF Model Transaction version 1.4.0 third-party software.

EMF Model Transaction version 1.4.0 is distributed under the terms of the Eclipse Public License version 1.0 and is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the about.html files for information about third-party software, licensing, and copyright information. These can be displayed from the Help menu item and selecting “About Inflexion UI Express”. You can view a copy of the Eclipse Public License version 1.0 at: `<install_root>/docs/legal/eclipse_1.0.pdf`. To obtain a copy of the Eclipse Modeling Tool source code, send a request to request_sourcecode@mentor.com.

- This software application may include EMF Validation Framework SDK version 1.4.0 third-party software.

EMF Validation Framework SDK version 1.4.0 is distributed under the terms of the Eclipse Public License version 1.0 and is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the about.html files for information about third-party software, licensing, and copyright information. These can be displayed from the Help menu item and selecting “About Inflexion UI Express”. You can view a copy of the Eclipse Public License version 1.0 at: `<install_root>/docs/legal/eclipse_1.0.pdf`. To obtain a copy of the Eclipse Modeling Tool source code, send a request to request_sourcecode@mentor.com.

- This software application may include Eclipse Graphical Editing Framework SDK version 3.6.0 third-party software.

Eclipse Graphical Editing Framework SDK version 3.6.0 is distributed under the terms of the Eclipse Public License version 1.0 and is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the about.html files for information about third-party software, licensing, and copyright information. These can be displayed from the Help menu item and selecting “About Inflexion UI Express”. You can view a copy of the Eclipse Public License version 1.0 at: `<install_root>/docs/legal/eclipse_1.0.pdf`. To obtain a copy of the Eclipse Modeling Tool source code, send a request to request_sourcecode@mentor.com.

- This software application may include Eclipse Web Tools Platform SDK version 3.2.0 third-party software.

Eclipse Web Tools Platform SDK version 3.2.0 is distributed under the terms of the Eclipse Public License version 1.0 and is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the about.html files for information about third-party software, licensing, and copyright information. These can be displayed from the Help menu item and selecting "About Inflexion UI Express". You can view a copy of the Eclipse Public License version 1.0 at: `<install_root>/legal/eclipse_1.0.pdf`. Portions of this software may be subject to the Apache License version 2.0. You can view a copy of the Apache License at `<install_root>/legal/apache_2.0.pdf`. Portions of this software may be subject to the Apache License version 1.1. You can view a copy of the Apache License at `<install_root>/legal/apache_1.1.pdf`. Portions of this software may be subject to the W3C Software Notice and License.

You can view a copy of the W3C Software License at [<install_root>/legal/w3c_2002.pdf](install_root/legal/w3c_2002.pdf). To obtain a copy of the Eclipse Web Tools Platform SDK version 3.2.0 source code, send a request to request_sourcecode@mentor.com.

- This software application may include NVIDIA Texture Tools version 2.0 third-party software, which is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. NVIDIA Texture Tools version 2.0 may be subject to the following copyrights:

© NVIDIA Corporation 2007 -- Ignacio Castano <icastano@nvidia.com>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the

Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

- This software application may include PNG version 1.4.5 third-party software, which is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. PNG version 1.4.5 may include the following copyright which is dual-licensed, and we elect "License 1" as shown below:

© 1998-2008 Greg Roelofs. All rights reserved.

This software is provided "as is," without warranty of any kind, express or implied. In no event shall the author or contributors be held liable for any damages arising in any way from the use of this software.

The contents of this file are DUAL-LICENSED. You may modify and/or redistribute this software according to the terms of one of the following two licenses (at your option):

LICENSE 1("BSD-like with advertising clause"):

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. Redistributions of source code must retain the above copyright notice, disclaimer, and this list of conditions.
2. Redistributions in binary form must reproduce the above copyright notice, disclaimer, and this list of conditions in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgment:

This product includes software developed by Greg Roelofs and contributors for the book, "PNG: The Definitive Guide," published by O'Reilly and Associates.

PNG version 1.4.5 may be subject to the following copyrights:

© 1999 by Willem van Schaik <willem@schaik.com>

version 1.0 - 1999.10.15 - First version.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this

permission notice appear in supporting documentation. This software is provided "as is" without express or implied warranty.

- This software application may include ZLIB version 1.2.5 third-party software.

ZLIB version 1.2.5 is distributed under the terms of the ZLIB version 1.2.5 and is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the license for the specific language governing rights and limitations under the license. You can view a copy of the license at: `<install_root>/docs/legal/boost_1.0.pdf`.

- This software application may include header files available from Khronos Group, which is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied.
- This software application may include extracts from Android version 18 Jan 2011 third-party software.

Android version 18 Jan 2011 is distributed under the terms of the Apache version 2.0 and is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the license for the specific language governing rights and limitations under the license. You can view a copy of the license at: `<install_root>/docs/legal/apache_2.0.pdf`. Portions of this software may be subject to Eclipse Public License version 1.0. You can view a copy of the Eclipse Public License version 1.0 at: `<install_root>/docs/legal/eclipse_1.0.pdf`.

- This software application may include extracts from Android version 18 Jan 2011 third-party software.

Android version 18 Jan 2011 is distributed under the terms of the Eclipse version 1.0 and is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the license for the specific language governing rights and limitations under the license. You can view a copy of the license at: `<install_root>/docs/legal/eclipse_1.0.pdf`. Portions of this software may be subject to Apache License version 2.0. You can view a copy of the Apache License version 2.0 at: `<install_root>/docs/legal/apache_2.0.pdf`.

- This software application may include FreeType version 2.3.12 third-party software.

FreeType version 2.3.12 is distributed under the terms of the FreeType Project License and is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the license for the specific language governing rights and limitations under the license. You can view a copy of the license at: `<install_root>/legal/freetype_2006.pdf`. Portions of this software may be subject to the Apache license version 2.0. You can view a copy of the license at: `<install_root>/legal/apache_2.0.pdf`.

- This software application may include Libpng (Android) version 1.2.44 third-party software, which is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. PNG version 1.4.5 may include the following copyright which is dual-licensed, and we elect "License 1" as shown below:

© 1998-2008 Greg Roelofs. All rights reserved.

This software is provided "as is," without warranty of any kind, express or implied. In no event shall the author or contributors be held liable for any damages arising in any way from the use of this software.

The contents of this file are DUAL-LICENSED. You may modify and/or redistribute this software according to the terms of one of the following two licenses (at your option):

LICENSE 1("BSD-like with advertising clause"):

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. Redistributions of source code must retain the above copyright notice, disclaimer, and this list of conditions.
2. Redistributions in binary form must reproduce the above copyright notice, disclaimer, and this list of conditions in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgment:

This product includes software developed by Greg Roelofs and contributors for the book, "PNG: The Definitive Guide," published by O'Reilly and Associates.

Libpng (Android) version 1.2.44 may be subject to the following copyrights:

© 1999 by Willem van Schaik <willem@schaik.com>

version 1.0 - 1999.10.15 - First version.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. This software is provided "as is" without express or implied warranty.

- This software application may include Zlib (Android) version 1.2.3 third-party software. Zlib (Android) version 1.2.3 is distributed under the terms of the BOOST Software License version 1.0 and is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the license for the specific language governing rights and limitations under the license. You can view a copy of the license at: <install_root>/docs/legal/boost_1.0.pdf.
- This software application may include Lua version 5.1.4 third-party software, which is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied.

This software application may include GNU FriBidi version 0.10.9 third-party software. GNU FriBidi version 0.10.9 is distributed under the terms of the GNU Lesser General Public License version 2.1 and is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the license for the specific language governing rights and limitations under the license. You can view a copy of the license at: <your_Mentor_Graphics_documentation_directory>/legal/gnu_lgpl_2.1.pdf. Portions of this software may be subject to the Unicode Terms of Use version 2009. You can view a copy of the Unicode Terms of Use version 2009 at: <your_Mentor_Graphics_documentation_directory>/legal/unicode_term_of_use_2009.pdf. To obtain a copy of the GNU FriBidi version 0.10.9 source code, send a request to request_sourcecode@mentor.com. This offer shall only be available for three years from the date Mentor Graphics Corporation first distributed GNU FriBidi version 0.10.9.

- This software application may include GNU FriBidi version 0.10.9 third-party software. GNU FriBidi version 0.10.9 is distributed under the terms of the GNU Lesser General Public License version 2.1 and is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the license for the specific language governing rights and limitations under the license. You can view a copy of the license at: <your_Mentor_Graphics_documentation_directory>/legal/gnu_lgpl_2.1.pdf. Portions of this software may be subject to the Unicode Terms of Use version 2009. You can view a copy of the Unicode Terms of Use version 2009 at: <your_Mentor_Graphics_documentation_directory>/legal/unicode_term_of_use_2009.pdf. To obtain a copy of the GNU FriBidi version 0.10.9 source code, send a request to request_sourcecode@mentor.com. This offer shall only be available for three years from the date Mentor Graphics Corporation first distributed GNU FriBidi version 0.10.9.
- This software application may include Harfbuzz version 0.9.0 third-party software, which is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied.
- This software application may include UCDN - Unicode Database and Normalization version Master, commit 92e82cddb6c11df5dc606388f314abec27c98e70 third-party software. UCDN - Unicode Database and Normalization version Master, commit 92e82cddb6c11df5dc606388f314abec27c98e70 is distributed under the terms of the PYTHON SOFTWARE FOUNDATION LICENSE version 2 and is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the license for the specific language governing rights and limitations under the license. You can view a copy of the license at: <your_Mentor_Graphics_documentation_directory>/legal/python_2.0.pdf. Portions of this software may be subject to the Unicode Terms of Use version 2009. You can view a copy of the Unicode Terms of Use version 2009 at: <your_Mentor_Graphics_documentation_directory>/legal/unicode_term_of_use_2009.pdf.
- This software application may include libjpeg version 8d third-party software, which is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. libjpeg version 8d may be subject to the following copyrights:

© 1989 by Jef Poskanzer.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. This software is provided "as is" without express or implied warranty.

© 1991-2012, Thomas G. Lane, Guido Vollbeding.

All Rights Reserved except as specified below.

Permission is hereby granted to use, copy, modify, and distribute this software (or portions thereof) for any purpose, without fee, subject to these conditions:

1. If any part of the source code for this software is distributed, then this README file must be included, with this copyright and no-warranty notice unaltered; and any additions, deletions, or changes to the original files must be clearly indicated in accompanying documentation.
 2. If only executable code is distributed, then the accompanying documentation must state that "this software is based in part on the work of the Independent JPEG Group".
 3. Permission for use of this software is granted only if the user accepts full responsibility for any undesirable consequences; the authors accept NO LIABILITY for damages of any kind.
- This software application may include lateef.ttf version 1.001 third-party software. lateef.ttf version 1.001 is distributed under the terms of the SIL Open Font License version 1.1 and is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the license for the specific language governing rights and limitations under the license. You can view a copy of the license at: <your_Mentor_Graphics_documentation_directory>/legal/sil_open_font_1.1.pdf.

Embedded Software and Hardware License Agreement

The latest version of the Embedded Software and Hardware License Agreement is available on-line at:
www.mentor.com/eshla

IMPORTANT INFORMATION

USE OF ALL PRODUCTS IS SUBJECT TO LICENSE RESTRICTIONS. CAREFULLY READ THIS LICENSE AGREEMENT BEFORE USING THE PRODUCTS. USE OF PRODUCTS INDICATES CUSTOMER'S COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT. ANY ADDITIONAL OR DIFFERENT PURCHASE ORDER TERMS AND CONDITIONS SHALL NOT APPLY.

EMBEDDED SOFTWARE AND HARDWARE LICENSE AGREEMENT ("Agreement")

This is a legal agreement concerning the use of Products (as defined in Section 1) between the company acquiring the Products ("Customer"), and the Mentor Graphics entity that issued the corresponding quotation or, if no quotation was issued, the applicable local Mentor Graphics entity ("Mentor Graphics"). Except for license agreements related to the subject matter of this license agreement which are physically signed by Customer and an authorized representative of Mentor Graphics, this Agreement and the applicable quotation contain the parties' entire understanding relating to the subject matter and supersede all prior or contemporaneous agreements. If Customer does not agree to these terms and conditions, promptly return or, in the case of Products received electronically, certify destruction of Products and all accompanying items within five days after receipt of such Products and receive a full refund of any license fee paid.

1. **Definitions.** As used in this Agreement and any applicable quotation, supplement, attachment and/or addendum ("Addenda"), these terms shall have the following meanings:
 - 1.1. "Customer's Product" means Customer's end-user product identified by a unique SKU (including any Related SKUs) in an applicable Addenda that is developed, manufactured, branded and shipped solely by Customer or an authorized manufacturer or subcontractor on behalf of Customer to end-users or consumers;
 - 1.2. "Developer" means a unique user, as identified by a unique user identification number, with access to Embedded Software at an authorized Development Location. A unique user is an individual who works directly with the embedded software in source code form, or creates, modifies or compiles software that ultimately links to the Embedded Software in Object Code form and is embedded into Customer's Product at the point of manufacture;
 - 1.3. "Development Location" means the location where Products may be used as authorized in the applicable Addenda;
 - 1.4. "Development Tools" means the software that may be used by Customer for building, editing, compiling, debugging or prototyping Customer's Product;
 - 1.5. "Embedded Software" means Software that is embeddable;
 - 1.6. "End-User" means Customer's customer;
 - 1.7. "Executable Code" means a compiled program translated into a machine-readable format that can be loaded into memory and run by a certain processor;
 - 1.8. "Hardware" means a physically tangible electro-mechanical system or sub-system and associated documentation;
 - 1.9. "Linkable Object Code" or "Object Code" means linkable code resulting from the translation, processing, or compiling of Source Code by a computer into machine-readable format;
 - 1.10. "Mentor Embedded Linux" or "MEL" means Mentor Graphics' tools, source code, and recipes for building Linux systems;
 - 1.11. "Open Source Software" or "OSS" means software subject to an open source license which requires as a condition for redistribution of such software, including modifications thereto, that the: (i) redistribution be in source code form or be made available in source code form; (ii) redistributed software be licensed to allow the making of derivative works; or (iii) redistribution be at no charge;
 - 1.12. "Processor" means the specific microprocessor to be used with Software and implemented in Customer's Product;
 - 1.13. "Products" means Software, Term-Licensed Products and/or Hardware;
 - 1.14. "Proprietary Components" means the components of the Products that are owned and/or licensed by Mentor Graphics and are not subject to an Open Source Software license, as more fully set forth in the product documentation provided with the Products;

- 1.15. “Redistributable Components” means those components that are intended to be incorporated or linked into Customer’s Linkable Object Code developed with the Software, as more fully set forth in the documentation provided with the Products;
- 1.16. “Related SKU” means two or more Customer Products identified by logically-related SKUs, where there is no difference or change in the electrical hardware or software content between such Customer Products;
- 1.17. “Software” means software programs, Embedded Software and/or Development Tools, including any updates, modifications, revisions, copies, documentation and design data that are licensed under this Agreement;
- 1.18. “Source Code” means software in a form in which the program logic is readily understandable by a human being;
- 1.19. “Sourcery CodeBench Software” means Mentor Graphics’ Development Tool for C/C++ embedded application development;
- 1.20. “Sourcery VSIPL++” is Software providing C++ classes and functions for writing embedded signal processing applications designed to run on one or more processors;
- 1.21. “Stock Keeping Unit” or “SKU” is a unique number or code used to identify each distinct product, item or service available for purchase;
- 1.22. “Subsidiary” means any corporation more than 50% owned by Customer, excluding Mentor Graphics competitors. Customer agrees to fulfill the obligations of such Subsidiary in the event of default. To the extent Mentor Graphics authorizes any Subsidiary’s use of Products under this Agreement, Customer agrees to ensure such Subsidiary’s compliance with the terms of this Agreement and will be liable for any breach by a Subsidiary; and
- 1.23. “Term-Licensed Products” means Products licensed to Customer for a limited time period (“Term”).

2. Orders, Fees and Payment.

- 2.1. To the extent Customer (or if agreed by Mentor Graphics, Customer’s appointed third party buying agent) places and Mentor Graphics accepts purchase orders pursuant to this Agreement (“Order(s)”), each Order will constitute a contract between Customer and Mentor Graphics, which shall be governed solely and exclusively by the terms and conditions of this Agreement and any applicable Addenda, whether or not these documents are referenced on the Order. Any additional or conflicting terms and conditions appearing on an Order will not be effective unless agreed in writing by an authorized representative of Customer and Mentor Graphics.
- 2.2. Amounts invoiced will be paid, in the currency specified on the applicable invoice, within 30 days from the date of such invoice. All invoices will be sent electronically to Customer on the date stated on the invoice unless otherwise specified in an Addendum. Any past due invoices will be subject to the imposition of interest charges in the amount of one and one-half percent per month or the applicable legal rate currently in effect, whichever is lower. Prices do not include freight, insurance, customs duties, taxes or other similar charges, which Mentor Graphics will state separately in the applicable invoice(s). Unless timely provided with a valid certificate of exemption or other evidence that items are not taxable, Mentor Graphics will invoice Customer for all applicable taxes including, but not limited to, VAT, GST, sales tax, consumption tax and service tax. Customer will make all payments free and clear of, and without reduction for, any withholding or other taxes; any such taxes imposed on payments by Customer hereunder will be Customer’s sole responsibility. If Customer appoints a third party to place purchase orders and/or make payments on Customer’s behalf, Customer shall be liable for payment under Orders placed by such third party in the event of default.
- 2.3. All Products are delivered FCA factory (Incoterms 2010), freight prepaid and invoiced to Customer, except Software delivered electronically, which shall be deemed delivered when made available to Customer for download. Mentor Graphics’ delivery of Software by electronic means is subject to Customer’s provision of both a primary and an alternate e-mail address.

3. Grant of License.

- 3.1. The Products installed, downloaded, or otherwise acquired by Customer under this Agreement constitute or contain copyrighted, trade secret, proprietary and confidential information of Mentor Graphics or its licensors, who maintain exclusive title to all Software and retain all rights not expressly granted by this Agreement. Mentor Graphics grants to Customer, subject to payment of applicable license fees, a nontransferable, nonexclusive license to use Software as described in the applicable Addenda. The limited licenses granted under the applicable Addenda shall continue until the expiration date of Term-Licensed Products or termination in accordance with Section 12 below, whichever occurs first. **Mentor Graphics does NOT grant Customer any right to (a) sublicense or (b) use Software beyond the scope of this Section without first signing a separate agreement or Addenda with Mentor Graphics for such purpose.**
- 3.2. License Type. The license type shall be identified in the applicable Addenda.
- 3.2.1. Development License: During the Term, if any, Customer may modify, compile, assemble and convert the applicable Embedded Software Source Code into Linkable Object Code and/or Executable Code form by the number of Developers specified, for the Processor(s), Customer’s Product(s) and at the Development Location(s) identified in the applicable Addenda.

- 3.2.2. End-User Product License: During the Term, if any, and unless otherwise specified in the applicable Addenda, Customer may incorporate or embed an Executable Code version of the Embedded Software into the specified number of copies of Customer's Product(s), using the Processor Unit(s), and at the Development Location(s) identified in the applicable Addenda. Customer may manufacture, brand and distribute such Customer's Product(s) worldwide to its End-Users.
- 3.2.3. Internal Tool License: During the Term, if any, Customer may use the Development Tools solely: (a) for internal business purposes and (b) on the specified number of computer work stations and sites. Development Tools are licensed on a per-seat or floating basis, as specified in the applicable Addenda, and shall not be distributed to others or delivered in Customer's Product(s) unless specifically authorized in an applicable Addenda.
- 3.2.4. Sourcery CodeBench Professional Edition License: During the Term specified in the applicable Addenda, Customer may (a) install and use the Proprietary Components of the Software (i) if the license is a node-locked license, by a single user who uses the Software on up to two machines provided that only one copy of the Software is in use at any one time, or (ii) if the license is a floating license, by the authorized number of concurrent users on one or more machines provided that only the authorized number of copies of the Software are in use at any one time, and (b) distribute the Redistributable Components of the Software in Executable Code form only and only as part of Customer's Object Code developed with the Software that provides substantially different functionality than the Redistributable Component(s) alone.
- 3.2.5. Sourcery CodeBench Standard Edition License: During the Term specified in the applicable Addenda, Customer may (a) install and use the Proprietary Components of the Software by a single user who uses the Software on up to two machines provided that only one copy of the Software is in use at any one time, and (b) distribute the Redistributable Component(s) of the Software in Executable Code form only and only as part of Customer's Object Code developed with the Software that provides substantially different functionality than the Redistributable Component(s) alone.
- 3.2.6. Sourcery CodeBench Personal Edition License: During the Term specified in the applicable Addenda, Customer may (a) install and use the Proprietary Components of the Software by a single user who uses the Software on one machine, and (b) distribute the Redistributable Component(s) of the Software in Executable Code form only and only as part of Customer Object Code developed with the Software that provides substantially different functionality than the Redistributable Component(s) alone.
- 3.2.7. Sourcery CodeBench Academic Edition License: During the Term specified in the applicable Addenda, Customer may (a) install and use the Proprietary Components of the Software for non-commercial, academic purposes only by a single user who uses the Software on one machine, and (b) distribute the Redistributable Component(s) of the Software in Executable Code form only and only as part of Customer Object Code developed with the Software that provides substantially different functionality than the Redistributable Component(s) alone.
- 3.3. Mentor Graphics may from time to time, in its sole discretion, lend Products to Customer. For each loan, Mentor Graphics will identify in writing the quantity and description of Software loaned, the authorized location and the Term of the loan. Mentor Graphics will grant to Customer a temporary license to use the loaned Software solely for Customer's internal evaluation in a non-production environment. Customer shall return to Mentor Graphics or delete and destroy loaned Software on or before the expiration of the loan Term. Customer will sign a certification of such deletion or destruction if requested by Mentor Graphics.

4. Beta Code.

- 4.1. Portions or all of certain Products may contain code for experimental testing and evaluation ("Beta Code"), which may not be used without Mentor Graphics' explicit authorization. Upon Mentor Graphics' authorization, Mentor Graphics grants to Customer a temporary, nontransferable, nonexclusive license for experimental use to test and evaluate the Beta Code without charge for a limited period of time specified by Mentor Graphics. This grant and Customer's use of the Beta Code shall not be construed as marketing or offering to sell a license to the Beta Code, which Mentor Graphics may choose not to release commercially in any form.
- 4.2. If Mentor Graphics authorizes Customer to use the Beta Code, Customer agrees to evaluate and test the Beta Code under normal conditions as directed by Mentor Graphics. Customer will contact Mentor Graphics periodically during Customer's use of the Beta Code to discuss any malfunctions or suggested improvements. Upon completion of Customer's evaluation and testing, Customer will send to Mentor Graphics a written evaluation of the Beta Code, including its strengths, weaknesses and recommended improvements.
- 4.3. Customer agrees to maintain Beta Code in confidence and shall restrict access to the Beta Code, including the methods and concepts utilized therein, solely to those employees and Customer location(s) authorized by Mentor Graphics to perform beta testing. Customer agrees that any written evaluations and all inventions, product improvements, modifications or developments that Mentor Graphics conceived or made during or subsequent to this Agreement, including those based partly or wholly on Customer's feedback, will be the exclusive property of Mentor Graphics. Mentor Graphics will have exclusive rights, title and interest in all such property. The provisions of this Subsection 4.3 shall survive termination of this Agreement.

5. Restrictions on Use.

- 5.1. Customer may copy Software only as reasonably necessary to support the authorized use, including archival and backup purposes. Each copy must include all notices and legends embedded in Software and affixed to its medium and container as received from Mentor Graphics. All copies shall remain the property of Mentor Graphics or its licensors. Except where embedded in Executable Code form in Customer's Product, Customer shall maintain a record of the number and location of all copies of Software, including copies merged with other software and products, and shall make those records available to Mentor Graphics upon request. Customer shall not make Products available in any form to any person other than Customer's employees, authorized manufacturers or authorized contractors, excluding Mentor Graphics competitors, whose job performance requires access and who are under obligations of confidentiality. Customer shall take appropriate action to protect the confidentiality of Products and ensure that any person permitted access does not disclose or use Products except as permitted by this Agreement. Customer shall give Mentor Graphics immediate written notice of any unauthorized disclosure or use of the Products as soon as Customer learns or becomes aware of such unauthorized disclosure or use.
- 5.2. Customer acknowledges that the Products provided hereunder may contain Source Code which is proprietary and its confidentiality is of the highest importance and value to Mentor Graphics. Customer acknowledges that Mentor Graphics may be seriously harmed if such Source Code is disclosed in violation of this Agreement. Except as otherwise permitted for purposes of interoperability as specified by applicable and mandatory local law, Customer shall not reverse-assemble, reverse-compile, reverse-engineer or in any way derive any Source Code from Products that are not provided in Source Code form. Except as embedded in Executable Code in Customer's Product and distributed in the ordinary course of business, in no event shall Customer provide Products to Mentor Graphics competitors. Log files, data files, rule files and script files generated by or for the Software (collectively "Files") constitute and/or include confidential information of Mentor Graphics. Customer may share Files with third parties, excluding Mentor Graphics competitors, provided that the confidentiality of such Files is protected by written agreement at least as well as Customer protects other information of a similar nature or importance, but in any case with at least reasonable care. Under no circumstances shall Customer use Products or allow their use for the purpose of developing, enhancing or marketing any product that is in any way competitive with Products, or disclose to any third party the results of, or information pertaining to, any benchmark.
- 5.3. Customer may not assign this Agreement or the rights and duties under it, or relocate, sublicense or otherwise transfer the Products, whether by operation of law or otherwise ("Attempted Transfer"), without Mentor Graphics' prior written consent, which shall not be unreasonably withheld, and payment of Mentor Graphics' then-current applicable relocation and/or transfer fees. Any Attempted Transfer without Mentor Graphics' prior written consent shall be a material breach of this Agreement and may, at Mentor Graphics' option, result in the immediate termination of the Agreement and/or the licenses granted under this Agreement. The terms of this Agreement, including without limitation the licensing and assignment provisions, shall be binding upon Customer's permitted successors in interest and assigns.
- 5.4. Notwithstanding any provision in an OSS license agreement applicable to a component of the Sourcery CodeBench Software that permits the redistribution of such component to a third party in Source Code or binary form, Customer may not use any Mentor Graphics trademark, whether registered or unregistered, in connection with such distribution, and may not recompile the Open Source Software components with the --with-pkgversion or --with-bugurl configuration options that embed Mentor Graphics' trademarks in the resulting binary.
- 5.5. The provisions of this Section 5 shall survive the termination of this Agreement.

6. Support Services.

- 6.1. Except as described in Sections 6.2, 6.3 and 6.4 below, and unless otherwise specified in any applicable Addenda to this Agreement, to the extent Customer purchases support services, Mentor Graphics will provide Customer updates and technical support for the number of Developers at the Development Location(s) for which support is purchased in accordance with Mentor Graphics' then-current End-User Software Support Terms located at <http://supportnet.mentor.com/about/legal/>.
- 6.2. To the extent Customer purchases support services for Sourcery CodeBench Software, support will be provided solely in accordance with the provisions of this Section 6.2. Mentor Graphics shall provide updates and technical support to Customer as described herein only on the condition that Customer uses the Executable Code form of the Sourcery CodeBench Software for internal use only and/or distributes the Redistributable Components in Executable Code form only (except as provided in a separate redistribution agreement with Mentor Graphics or as required by the applicable Open Source license). Any other distribution by Customer of the Sourcery CodeBench Software (or any component thereof) in any form, including distribution permitted by the applicable Open Source license, shall automatically terminate any remaining support term. Subject to the foregoing and the payment of support fees, Mentor Graphics will provide Customer updates and technical support for the number of Developers at the Development Location(s) for which support is purchased in accordance with Mentor Graphics' then-current Sourcery CodeBench Software Support Terms located at <http://www.mentor.com/codebench-support-legal>.
- 6.3. To the extent Customer purchases support services for Sourcery VSIPL++, Mentor Graphics will provide Customer updates and technical support for the number of Developers at the Development Location(s) for which support is purchased solely in accordance with Mentor Graphics' then-current Sourcery VSIPL++ Support Terms located at <http://www.mentor.com/vsipl-support-legal>.
- 6.4. To the extent Customer purchases support services for Mentor Embedded Linux, Mentor Graphics will provide Customer updates and technical support for the number of Developers at the Development Location(s) for which support is purchased

solely in accordance with Mentor Graphics' then-current Mentor Embedded Linux Support Terms located at <http://www.mentor.com/mel-support-legal>.

7. **Third Party and Open Source Software.** Products may contain Open Source Software or code distributed under a proprietary third party license agreement. Please see applicable Products documentation, including but not limited to license notice files, header files or source code for further details. Please see the applicable Open Source Software license(s) for additional rights and obligations governing your use and distribution of Open Source Software. Customer agrees that it shall not subject any Product provided by Mentor Graphics under this Agreement to any Open Source Software license that does not otherwise apply to such Product. In the event of conflict between the terms of this Agreement, any Addenda and an applicable OSS or proprietary third party agreement, the OSS or proprietary third party agreement will control solely with respect to the OSS or proprietary third party software component. The provisions of this Section 7 shall survive the termination of this Agreement.
8. **Limited Warranty.**
 - 8.1. Mentor Graphics warrants that during the warranty period its standard, generally supported Products, when properly installed, will substantially conform to the functional specifications set forth in the applicable user manual and/or specification. Mentor Graphics does not warrant that Products will meet Customer's requirements or that operation of Products will be uninterrupted or error free. The warranty period is 90 days starting on the 15th day after delivery or upon installation, whichever first occurs. Customer must notify Mentor Graphics in writing of any nonconformity within the warranty period. For the avoidance of doubt, this warranty applies only to the initial shipment of Products under an Order and does not renew or reset, for example, with the delivery of (a) Software updates or (b) authorization codes. This warranty shall not be valid if Products have been subject to misuse, unauthorized modification or improper installation. MENTOR GRAPHICS' ENTIRE LIABILITY AND CUSTOMER'S EXCLUSIVE REMEDY SHALL BE, AT MENTOR GRAPHICS' OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF THE PRODUCTS TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF THE PRODUCTS THAT DO NOT MEET THIS LIMITED WARRANTY, PROVIDED CUSTOMER HAS OTHERWISE COMPLIED WITH THIS AGREEMENT. MENTOR GRAPHICS MAKES NO WARRANTIES WITH RESPECT TO: (A) SERVICES; OR (B) PRODUCTS PROVIDED AT NO CHARGE, WHICH ARE PROVIDED "AS IS" UNLESS OTHERWISE AGREED IN WRITING.
 - 8.2. THE WARRANTIES SET FORTH IN THIS SECTION 8 ARE EXCLUSIVE TO CUSTOMER AND DO NOT APPLY TO ANY END-USER. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS MAKE ANY OTHER WARRANTIES, EXPRESS, IMPLIED, OR STATUTORY, WITH RESPECT TO PRODUCTS OR OTHER MATERIAL PROVIDED UNDER THIS AGREEMENT. MENTOR GRAPHICS AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.
9. **LIMITATION OF LIABILITY.** EXCEPT WHERE THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, AND EXCEPT FOR EITHER PARTY'S BREACH OF ITS CONFIDENTIALITY OBLIGATIONS, CUSTOMER'S BREACH OF LICENSING TERMS OR CUSTOMER'S OBLIGATIONS UNDER SECTION 10, IN NO EVENT SHALL: (A) EITHER PARTY OR ITS RESPECTIVE LICENSORS BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS OR SAVINGS) WHETHER BASED ON CONTRACT, TORT OR ANY OTHER LEGAL THEORY, EVEN IF SUCH PARTY OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES; AND (B) EITHER PARTY OR ITS RESPECTIVE LICENSORS' LIABILITY UNDER THIS AGREEMENT, INCLUDING, FOR THE AVOIDANCE OF DOUBT, LIABILITY FOR ATTORNEYS' FEES OR COSTS, EXCEED THE GREATER OF THE FEES PAID OR OWING TO MENTOR GRAPHICS FOR THE PRODUCT OR SERVICE GIVING RISE TO THE CLAIM OR \$500,000 (FIVE HUNDRED THOUSAND U.S. DOLLARS). NOTWITHSTANDING THE FOREGOING, IN THE CASE WHERE NO AMOUNT WAS PAID, MENTOR GRAPHICS AND ITS LICENSORS SHALL HAVE NO LIABILITY FOR ANY DAMAGES WHATSOEVER. THE PROVISIONS OF THIS SECTION 9 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.
10. **Hazardous Applications.**
 - 10.1. Customer agrees that Mentor Graphics has no control over Customer's testing or the specific applications and use that Customer will make of Products. Mentor Graphics Products are not specifically designed for use in the operation of nuclear facilities, aircraft navigation or communications systems, air traffic control, life support systems, medical devices or other applications in which the failure of Mentor Graphics Products could lead to death, personal injury, or severe physical or environmental damage ("Hazardous Applications").
 - 10.2. CUSTOMER ACKNOWLEDGES IT IS SOLELY RESPONSIBLE FOR TESTING PRODUCTS USED IN HAZARDOUS APPLICATIONS AND SHALL BE SOLELY LIABLE FOR ANY DAMAGES RESULTING FROM SUCH USE. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS SHALL BE LIABLE FOR ANY DAMAGES RESULTING FROM OR IN CONNECTION WITH THE USE OF PRODUCTS IN ANY HAZARDOUS APPLICATIONS.
 - 10.3. CUSTOMER AGREES TO INDEMNIFY AND HOLD HARMLESS MENTOR GRAPHICS AND ITS LICENSORS FROM ANY CLAIMS, LOSS, COST, DAMAGE, EXPENSE OR LIABILITY, INCLUDING REASONABLE ATTORNEYS' FEES, ARISING OUT OF OR IN CONNECTION WITH THE USE OF PRODUCTS AS DESCRIBED IN SECTION 10.1.
 - 10.4. THE PROVISIONS OF THIS SECTION 10 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

11. Infringement.

- 11.1. Mentor Graphics will defend or settle, at its option and expense, any action brought against Customer in the United States, Canada, Japan, or member state of the European Union which alleges that any standard, generally supported Product acquired by Customer hereunder infringes a patent or copyright or misappropriates a trade secret in such jurisdiction. Mentor Graphics will pay any costs and damages finally awarded against Customer that are attributable to the action. Customer understands and agrees that as conditions to Mentor Graphics' obligations under this section Customer must: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance to settle or defend the action; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the action.
 - 11.2. If a claim is made under Subsection 11.1 Mentor Graphics may, at its option and expense, and in addition to its obligations under Section 11.1, either (a) replace or modify the Product so that it becomes noninfringing; or (b) procure for Customer the right to continue using the Product. If Mentor Graphics determines that neither of those alternatives is financially practical or otherwise reasonably available, Mentor Graphics may require the return of the Product and refund to Customer any purchase price or license fee(s) paid.
 - 11.3. Mentor Graphics has no liability to Customer if the claim is based upon: (a) the combination of the Product with any product not furnished by Mentor Graphics, where the Product itself is not infringing; (b) the modification of the Product other than by Mentor Graphics or as directed by Mentor Graphics, where the unmodified Product would not infringe; (c) the use of the infringing Product when Mentor Graphics has provided Customer with a current unaltered release of a non-infringing Product of substantially similar functionality in accordance with Subsection 11.2(a); (d) the use of the Product as part of an infringing process; (e) a product that Customer makes, uses, or sells, where the Product itself is not infringing; (f) any Product provided at no charge; (g) any software provided by Mentor Graphics' licensors who do not provide such indemnification to Mentor Graphics' customers; (h) Open Source Software, except to the extent that the infringement is directly caused by Mentor Graphics' modifications to such Open Source Software; or (i) infringement by Customer that is deemed willful. In the case of (i), Customer shall reimburse Mentor Graphics for its reasonable attorneys' fees and other costs related to the action.
 - 11.4. THIS SECTION 11 IS SUBJECT TO SECTION 9 ABOVE AND STATES: (A) THE ENTIRE LIABILITY OF MENTOR GRAPHICS AND ITS LICENSORS AND (B) CUSTOMER'S SOLE AND EXCLUSIVE REMEDY, WITH RESPECT TO ANY ALLEGED PATENT OR COPYRIGHT INFRINGEMENT OR TRADE SECRET MISAPPROPRIATION BY ANY PRODUCT PROVIDED UNDER THIS AGREEMENT.
12. **Termination and Effect of Termination.** If a Software license was provided for limited term use, such license will automatically terminate at the end of the authorized Term.
- 12.1. Termination for Breach. This Agreement shall remain in effect until terminated in accordance with its terms. Mentor Graphics may terminate this Agreement and/or any licenses granted under this Agreement, and Customer will immediately discontinue use and distribution of Products, if Customer (a) commits any material breach of any provision of this Agreement and fails to cure such breach upon 30-days prior written notice; or (b) becomes insolvent, files a bankruptcy petition, institutes proceedings for liquidation or winding up or enters into an agreement to assign its assets for the benefit of creditors. Termination of this Agreement or any license granted hereunder will not affect Customer's obligation to pay for Products shipped or licenses granted prior to the termination, which amounts shall be payable immediately upon the date of termination. For the avoidance of doubt, nothing in this Section 12 shall be construed to prevent Mentor Graphics from seeking immediate injunctive relief in the event of any threatened or actual breach of Customer's obligations hereunder.
 - 12.2. Effect of Termination. Upon termination of this Agreement, the rights and obligations of the parties shall cease except as expressly set forth in this Agreement. Upon termination or expiration of the Term, Customer will discontinue use and/or distribution of Products, and shall return Hardware and either return to Mentor Graphics or destroy Software in Customer's possession, including all copies and documentation, and certify in writing to Mentor Graphics within ten business days of the termination date that Customer no longer possesses any of the affected Products or copies of Software in any form, except to the extent an Open Source Software license conflicts with this Section 12.2 and permits Customer's continued use of any Open Source Software portion or component of a Product. Upon termination for Customer's breach, an End-User may continue its use and/or distribution of Customer's Product so long as: (a) the End-User was licensed according to the terms of this Agreement, if applicable to such End-User, and (b) such End-User is not in breach of its agreement, if applicable, nor a party to Customer's breach.
13. **Export.** The Products provided hereunder are subject to regulation by local laws and United States government agencies, which prohibit export or diversion of certain products, information about the products, and direct or indirect products thereof, to certain countries and certain persons. Customer agrees that it will not export Products in any manner without first obtaining all necessary approval from appropriate local and United States government agencies. Customer acknowledges that the regulation of product export is in continuous modification by local governments and/or the United States Congress and administrative agencies. Customer agrees to complete all documents and to meet all requirements arising out of such modifications.
14. **U.S. Government License Rights.** Software was developed entirely at private expense. All Software is commercial computer software within the meaning of the applicable acquisition regulations. Accordingly, pursuant to US FAR 48 CFR 12.212 and DFAR 48 CFR 227.7202, use, duplication and disclosure of the Software by or for the U.S. Government or a U.S. Government subcontractor is subject solely to the terms and conditions set forth in this Agreement, except for provisions which are contrary to applicable mandatory federal laws.

15. **Third Party Beneficiary.** For any Products licensed under this Agreement and provided by Customer to End-Users, Mentor Graphics or the applicable licensor is a third party beneficiary of the agreement between Customer and End-User. Mentor Graphics Corporation, Mentor Graphics (Ireland) Limited, and other licensors may be third party beneficiaries of this Agreement with the right to enforce the obligations set forth herein.
16. **Review of License Usage.** Customer will monitor the access to and use of Software. With prior written notice, during Customer's normal business hours, and no more frequently than once per calendar year, Mentor Graphics may engage an internationally recognized accounting firm to review Customer's software monitoring system, records, accounts and sublicensing documents deemed relevant by the internationally recognized accounting firm to confirm Customer's compliance with the terms of this Agreement or U.S. or other local export laws. Such review may include FlexNet (or successor product) report log files that Customer shall capture and provide at Mentor Graphics' request. Customer shall make records available in electronic format and shall fully cooperate with data gathering to support the license review. Mentor Graphics shall bear the expense of any such review unless a material non-compliance is revealed. Mentor Graphics shall treat as confidential information all Customer information gained as a result of any request or review and shall only use or disclose such information as required by law or to enforce its rights under this Agreement. Such license review shall be at Mentor Graphics' expense unless it reveals a material underpayment of fees of five percent or more, in which case Customer shall reimburse Mentor Graphics for the costs of such license review. Customer shall promptly pay any such fees. If the license review reveals that Customer has made an overpayment, Mentor Graphics has the option to either provide the Customer with a refund or credit the amount overpaid to Customer's next payment. The provisions of this Section 16 shall survive the termination of this Agreement.
17. **Controlling Law, Jurisdiction and Dispute Resolution.** This Agreement shall be governed by and construed under the laws of the State of California, USA, excluding choice of law rules. All disputes arising out of or in relation to this Agreement shall be submitted to the exclusive jurisdiction of the state and federal courts of California, USA. Nothing in this section shall restrict Mentor Graphics' right to bring an action (including for example a motion for injunctive relief) against Customer or its Subsidiary in the jurisdiction where Customer's or its Subsidiary's place of business is located. The United Nations Convention on Contracts for the International Sale of Goods does not apply to this Agreement.
18. **Severability.** If any provision of this Agreement is held by a court of competent jurisdiction to be void, invalid, unenforceable or illegal, such provision shall be severed from this Agreement and the remaining provisions will remain in full force and effect.
19. **Miscellaneous.** This Agreement contains the parties' entire understanding relating to its subject matter and supersedes all prior or contemporaneous agreements, including but not limited to any purchase order terms and conditions. This Agreement may only be modified in writing, signed by an authorized representative of each party. Waiver of terms or excuse of breach must be in writing and shall not constitute subsequent consent, waiver or excuse.