JModule Integration User Manual

Version 0.1

(Preliminary and subject to change without notice)



Copyright © 2019 Bose Corporation. All Rights Reserved.

Contents

[Version History 3](#_Toc527119630)

[1 Introduction 4](#_Toc527119631)

[2 Initialization 4](#_Toc527119632)

[3 Periodic Processing Execution 4](#_Toc527119633)

[4 Control Command Handling 4](#_Toc527119634)

[5 Event Notification 4](#_Toc527119635)

[6 System Requirements 4](#_Toc527119636)

[Appendix 1: Bose Audio Framework API Function Reference 6](#_Toc527119637)

[Appendix 2: BAF API Integration Example Code 7](#_Toc527119638)

# Version History

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Comment** |
| 0.1 | Jan-16-2019 | Initial version |

# Introduction

The JModule is a generic algorithm software interface, which allows multiple algorithms to coexist in an OpenJADE Framework. OpenJADE provides the specifications and examples required to implement a JModule.

The JModule is defined by two major components:

* Description of the algorithm processes and their input/output ports
* Adapter, responsible for implementing the algorithm adaptation methods

The JModule\_create API sets the methods covering four aspects of an algorithm:

* Initialization
* Periodic processing execution
* Control command handling
* Notification

# Initialization

The initialization API is responsible for initializing internal data structures of the algorithm.

This API adapts the algorithm initialization APIs to the JModule initialization interface. For example, for a Simulink “C” generated algorithm, the initialization API has a prototype in the form void <Algorithm Name>\_initialize(void).

# Periodic Processing Execution

An algorithm may have one or more periodic processing execution. A periodic process may have as any number of input and/or output ports.

# Control Command Handling

An algorithm may have one control command handling interface. The JModule provides a generic interface that will pass a request message to the algorithm and returns a corresponding response.

# Event Notification

The JModule can support asynchronous notifications, as well as synchronous notification through polling. A set of three APIs are provided to registering callback, polling, and to invoking notifications. Depending on algorithm, only some of the methods will be implemented.

# 

# Appendix 1: JModule API Function Reference

JModule 3

JModule diagram 3

JModule 3

Processes 5

Process 5

exec diagram 6

Connection 6

Module 7

Ports 8

Port 9

JModule\_Func 10

JModule\_Bitfield 11

Port\_Direction 11

Port\_Data 12

Port\_Error 12

# JModule

Package in package 'Components'

## JModule diagram

This diagram shows the composition of the JModule component



1. JModule

## JModule

This is the top structure of a JModule component. The JModule component contains the classes/structures required for defining a JModule in a JModule Library. Within structure declarations, whenever possible, Variable Length Array (VLA) are used instead of Array of Pointers. VLA has the advantage of faster access, but the restriction of being the last member in the structure. VLA are allocated, most conveniently through C structure initializers.

| ATTRIBUTES |
| --- |
| id : \_JModule\_Id Public  Unique identifier listed in the JModule\_Id enumeration. |
| create : JModule\_Func\_Create Public  Pointer to the JModule creation API. This API should assign APIs to the init, command, register\_notify and the exec for each process pointers to functions. This API should be exposed to the JModule\_Desc file. |
| init : JModule\_Func\_Init Public  Pointer to the feature initialization API. This API should call algorithm specific initialization APIs and possibly initialize the state of the feature. |
| command : JModule\_Func\_Command Public  Pointer to the command sending API |
| register\_notify : JModule\_Func\_Register\_Notify Public  Pointer to the API that registers a notification function with the algorithm. |
| notify : JModule\_Func\_Notify Public  Pointer to the API that retrieves a notification from the JModule. |
| poll : JModule\_Func\_Poll Public  Pointer to the API that checks if JModule notifications are present. |
| processes : Processes\* Public  Pointer to the feature's Processes module |
| state\_size : uint32\_t Public  Size, in bytes, of the state structure. A memory buffer of this size will be allocated in the JModule\_Context structure for storing local states. |
| param\_size : uint32\_t Public  Size, in bytes, of the parameter structure. A memory buffer of this size will be allocated in the JModule\_Context structure for storing local parameters. |
| param : void\* Public  Pointer to the feature parameter structure. It is declared as void\* so the type can be overloaded by the JModule Developer. Set to NULL if the feature has no parameters, but param\_size can be set to a non-zero value, so local parameters can be stored in the JModule\_Context |
| state : void\* Public  Pointer to the feature state structure. It is declared as void\* so the type can be overloaded by the JModule Developer. Set to NULL if the feature has no states, but state\_size can be set to a non-zero value, so local parameters can be stored in the JModule\_Context |

| OPERATIONS |
| --- |
| create (this : JModule\* ) : void Public  API for invoking feature's pointer to the create function. |

## Processes

This structure is a container of the JModule processes. A feature may contain any number of processes, which can run at different thread priorities.

| ATTRIBUTES |
| --- |
| num\_items : int32\_t Public  Number of Process items in the Processes container |
| process : Process Public  Variable length array (VLA) of Process items. |

## Process

This class contains data structures and API required for executing a JModule Process. A process may operate on any number of input and/or output ports.

| ATTRIBUTES |
| --- |
| exec : JModule\_Func\_Process Public  Pointer to the process execution function. The JModule Developer implements the Algorithm wrapper API, which will be assigned to this pointer to function, i.e. registered with the Process. |
| call\_interval : uint32\_t Public  For synchronous processes, specifies the calling interval, in samples for the current process. |
| ports\_in : Ports\* Public  Pointer to the input Ports container |
| ports\_out : Ports\* Public  Pointer to the output Ports container |
| feature : void\* Public  Pointer to a feature context that will be used by the process API to access JModule states or parameters. |

| OPERATIONS |
| --- |
| exec (this : Process\* , context : Process\_Context\* , thread\_level : uint32\_t ) : void Public  API that invokes the exec pointer to function. The API first checks the availability of the buffers associated with the input/output Ports, and makes these buffers available after the exec pointer to function execution. |
| get\_port\_in (this : Process\* , port\_index : uint32\_t ) : Port\* Public  API that returns pointer to an input port specified by its index. The API returns NULL if the index is out of range. |
| get\_port\_out (this : Process\* , port\_index : uint32\_t ) : Port\* Public  API that returns pointer to an output port specified by its index. The API returns NULL if the index is out of range. |

### **Connection**

| ATTRIBUTES |
| --- |
| id : uint32\_t Public |
| size : uint32\_t Public |
| in\_start : uint32\_t Public |
| out\_start : uint32\_t Public |
| full : Semaphore Public |
| empty : Semaphore Public |
| in\_count : uint32\_t Public |
| out\_count : uint32\_t Public |

| OPERATIONS |
| --- |
| create (this : Connection\* , in\_start : uint32\_t , out\_start : uint32\_t , size : uint32\_t ) : bool Public |
| reset (this : Connection\* ) : void Public |
| alloc (this : Connection\* , buf[] : void\* ) : void Public |
| receive (this : Connection\* , buf[] : void\* ) : void Public |
| send (this : Connection\* ) : void Public |
| free (this : Connection\* ) : void Public |
| fill (this : Connection\* ) : void Public |
| empty (this : Connection\* ) : void Public |

### **Module**

| ATTRIBUTES |
| --- |
| id : Module\_Id Public |
| feature : JModule\* Public |
| context : JModule\_Context\* Public |

| OPERATIONS |
| --- |
| get\_process (this : Module\* , process\_index : uint32\_t ) : Process\* Public |
| get\_process\_context (this : Module\* , process\_index : uint32\_t ) : Process\_Context\* Public |
| set\_port\_in\_connection (this : Module\* , process\_index : uint32\_t , port\_index : uint32\_t , connection : Connection\* ) : Connection\* Public |
| set\_port\_out\_connection (this : Module\* , process\_index : uint32\_t , port\_index : uint32\_t , connection : Connection\* ) : Connection\* Public |

## Ports

This structure is a container of Port items.

| ATTRIBUTES |
| --- |
| num\_items : uint32\_t Public  Number of Port items in the Ports container |
| port : Port Public  Variable length array (VLA) of Port items. |

## Port

Data structure of Port properties and helper APIs. A port is connected to a sample buffer, but the pointer to the buffer is managed by a Port\_Context associated with a JModule instance. The same buffer can be shared by multiple ports, allowing for buffer reuse or for ports to be connected within a single threaded processes execution.

| CONNECTORS |
| --- |
| Dependency Source -> Destination  From: Port : Class, Public  To: Port\_Error : Enumeration, Public |
| ATTRIBUTES |
| data\_type : Port\_Data Public  Data type of the samples in the Port buffer . Valid values are one of the Port\_Data enumeration values |
| direction : Port\_Direction Public  Specify if this is an input or output port |
| frame\_size : uint32\_t Public  Size, in samples, of the processing frame. |
| num\_chan : uint32\_t Public  Number of channels in the sample buffer |
| sample\_rate : uint32\_t Public  Sample rate, in Hz |
| sample\_size : uint32\_t Public  Sample size, in bytes |
| block\_size : uint32\_t Public  Buffer frame size, in bytes = frame\_size\*num\_chan\*sample\_size. It is provided as an optimization convenience, for avoiding recalculating the buffer frame size. |
| passthru\_port : uint32\_t Public  The id plus one of the input port in the same process that can be used for pass thru operations to the current output port. Also the current input/output port will copy the parameters the output/input port in the same process, with the id specified. A value of zero means no pass thru or parameter copy port is specified. |

| OPERATIONS |
| --- |
| compare (this : Port\* , p : Port\* ) : Port\_Error Public  Helper API for comparing if two ports are compatible, so they can be connected. |
| compare\_merge (this : Port\* , p : Port\* ) : Port\_Error Public  Helper API for comparing if two ports are compatible, so they can be connected. It tries to merge the num\_chan and frame size of the two ports, if one of them is set to any (i.e. zero) |
| setup (this : Port\* , frame\_size : uint32\_t , num\_chan : uint32\_t , sample\_size : uint32\_t , sample\_rate : uint32\_t , data\_type : Port\_Data ) : bool Public  Set the port properties |
| block\_size (this : Port\* ) : uint32\_t Public  API for computing the frame size in bytes = frame\_size\*num\_chan\*sample\_size. |

## JModule\_Func

Type definitions for pointer to functions used by structures in the JModule component.

| ATTRIBUTES |
| --- |
| func : void\* Public |

| OPERATIONS |
| --- |
| JModule\_Func\_Create () : void Public  Pointer to feature creation API |
| JModule\_Func\_Init () : void Public  Pointer to JModule initialization API |
| JModule\_Func\_Process () : void Public  Pointer to Process execution API |
| JModule\_Func\_Notify () : void Public  Pointer to notification callback |
| JModule\_Func\_Command () : void Public  Pointer to command sender API |
| JModule\_Func\_Register\_Notify () : void Public  Pointer to notification registration API |
| JModule\_Func\_Poll () : void Public |

## JModule\_Bitfield

Enumeration and APIs to set/get JModule IDs as bitfields in a 32bit word

| ATTRIBUTES |
| --- |
| JMODULE\_NUM\_BIT : Public = 8  Number of bits used for the JModule ID |
| PROCESS\_NUM\_BIT : Public = 4  Number of bits used for the Process ID |
| PORT\_NUM\_BIT : Public = 4  Number of bits used for the Port ID |
| CONTEXT\_NUM\_BIT : Public = 8  Number of bits used for the JModule Context ID |
| DIRECTION\_NUM\_BIT : Public = 1  Number of bits used for specifying the port direction |

## Port\_Direction

Enumeration of possible port direction

| ATTRIBUTES |
| --- |
| PORT\_DIRECTION\_IN : Public  Input port |
| PORT\_DIRECTION\_OUT : Public  Output port |
| PORT\_DIRECTION\_IO : Public  Bidirectional port |

## Port\_Data

Enumeration of the port data types

| ATTRIBUTES |
| --- |
| PORT\_DATA\_INT32 : Public  32-bit signed integer |
| PORT\_DATA\_REAL32 : Public  32-bit single precision float |
| PORT\_DATA\_INT24 : Public  24-bit integer |

## Port\_Error

Enumeration of possible port mismatching errors

| CONNECTORS |
| --- |
| Dependency Source -> Destination  From: Port : Class, Public  To: Port\_Error : Enumeration, Public |
| ATTRIBUTES |
| PORT\_SUCCESS : Public  Success |
| PORT\_ERROR\_FRAME\_SIZE : Public  Different Frame Size |
| PORT\_ERROR\_SAMPLE\_SIZE : Public  Different Sample Size |
| PORT\_ERROR\_SAMPLING\_RATE : Public  Different sampling rate |
| PORT\_ERROR\_DATA\_TYPE : Public  Different data type |
| PORT\_ERROR\_FORMAT : Public  Different formatting |
| PORT\_ERROR\_DIRECTION : Public  Can connect only in/out |

# Appendix 2: JModule Example Code

## Example of JModule description file

|  |
| --- |
| #include "JModule.h"  #include "JModule\_Id.h"  #include "JModule\_Bitfield.h"  #include "JModule\_Desc\_Demoalg.h"  static Ports JModule\_process0\_port\_in = {  .num\_items = 1,  {{ // in  .data\_type = PORT\_DATA\_REAL32,  .direction = PORT\_DIRECTION\_IN,  .frame\_size = 32,  .num\_chan = 4,  .sample\_rate = 44100,  .sample\_size = sizeof(float)  }}  };  static Ports JModule\_process0\_port\_out = {  .num\_items = 1,  {{  .data\_type = PORT\_DATA\_REAL32,  .direction = PORT\_DIRECTION\_OUT,  .frame\_size = 32,  .num\_chan = 4,  .sample\_rate = 44100,  .sample\_size = sizeof(float)  }}  };  static Processes JModule\_processes = {  .num\_items = JMODULE\_DEMOALG\_NUM\_PROCESSES,  {{  .call\_interval = 1,  .ports\_in = &JModule\_process0\_port\_in,  .ports\_out = &JModule\_process0\_port\_out  }}  };  /\* Parameter Data block \*/  static JModule\_Demoalg\_State JModule\_State\_obj;  static JModule\_Demoalg\_Param JModule\_Param\_obj;  extern void JModule\_Adapt\_Demoalg\_create();  JModule JModule\_Demoalg\_obj = {  .id = JMODULE\_DEMOALG,  .processes = &JModule\_processes,  .create = JModule\_Adapt\_Demoalg\_create,  .state = &JModule\_State\_obj,  .param = &JModule\_Param\_obj  }; |

## JModule Algorithm Adapter Example

|  |
| --- |
| #include "JModule\_Adapt\_Demoalg.h"  /\* Model entry point functions \*/  extern void Demoalg\_initialize(void);  static void JModule\_Adapt\_init(JModule\* this, Processes\* processes, Processes\_Context\* context)  {  Demoalg\_initialize();  }  extern int Demoalg\_HandleAsdIdRequest(void\* pReqBuf, uint32\_t reqBufSize,  void\* pRspBuf, uint32\_t rspBufSize);    static void JModule\_Adapt\_send(JModule\* this, Processes\_Context\* context, Message\* req, Message\* rsp)  {  void \* req\_msg = req->message\_data;  void \* rsp\_msg = rsp->message\_data;  // Send message to the algorithm  Demoalg\_HandleAsdIdRequest( req->message\_data, 16, rsp->message\_data, 16 );  }  /\* External inputs (root inport signals with auto storage) \*/  typedef struct {  float In1[128]; /\* '<Root>/In1' \*/  } ExtU\_Demoalg\_T;  /\* External outputs (root outports fed by signals with auto storage) \*/  typedef struct {  float Out1[128]; /\* '<Root>/Out1' \*/  } ExtY\_Demoalg\_T;  /\* External inputs (root inport signals with auto storage) \*/  extern ExtU\_Demoalg\_T Demoalg\_U;  /\* External outputs (root outports fed by signals with auto storage) \*/  extern ExtY\_Demoalg\_T Demoalg\_Y;  extern void Demoalg\_step(void);  static void JModule\_Adapt\_process0(JModule\* this, Process\* process, Process\_Context\* process\_ctx, uint32\_t thread\_level)  {  Ports \*port\_in = process->ports\_in;  Ports \*port\_out = process->ports\_out;  Ports\_Context \*port\_in\_ctx = process\_ctx->ports\_in;  Ports\_Context \*port\_out\_ctx = process\_ctx->ports\_out;  uint32\_t i;  float \* src = port\_in\_ctx->item[0].payload;  float \* dst = port\_out\_ctx->item[0].payload;  for(i = 0; i < port\_in->port[0].frame\_size \* port\_in->port[0].num\_chan ; i++ ){  Demoalg\_U.In1[i] = src[i];  }  Demoalg\_step();  for(i = 0; i < port\_out->port[0].frame\_size \* port\_out->port[0].num\_chan ; i++ ){  dst[i] = Demoalg\_Y.Out1[i];  }    }  static void JModule\_Adapt\_register\_notify(JModule\* this, JModule\_Func\_Notify notify)  {    }  static void JModule\_Adapt\_poll( JModule\* this )  {    }  static void JModule\_Adapt\_notify( JModule\* this )  {    }  void JModule\_Adapt\_Demoalg\_create(JModule\* this)  {  this->processes->process[0].exec = JModule\_Adapt\_process0;  this->init = JModule\_Adapt\_init;  this->command = JModule\_Adapt\_send;  this->register\_notify = JModule\_Adapt\_register\_notify;  this->poll = JModule\_Adapt\_poll;  this->notify = JModule\_Adapt\_notify;  } |

## JModule Folder structure example:

|  |  |
| --- | --- |
| **JModule**  **├───build**  **├───lib**  **├───repo**  **└───src** | **JModule root folder**  **Framework build scripts**  **Algorithm library**  **Algorithm repository (Simulink Model)**  **Algorithm source files from which lib is built** |