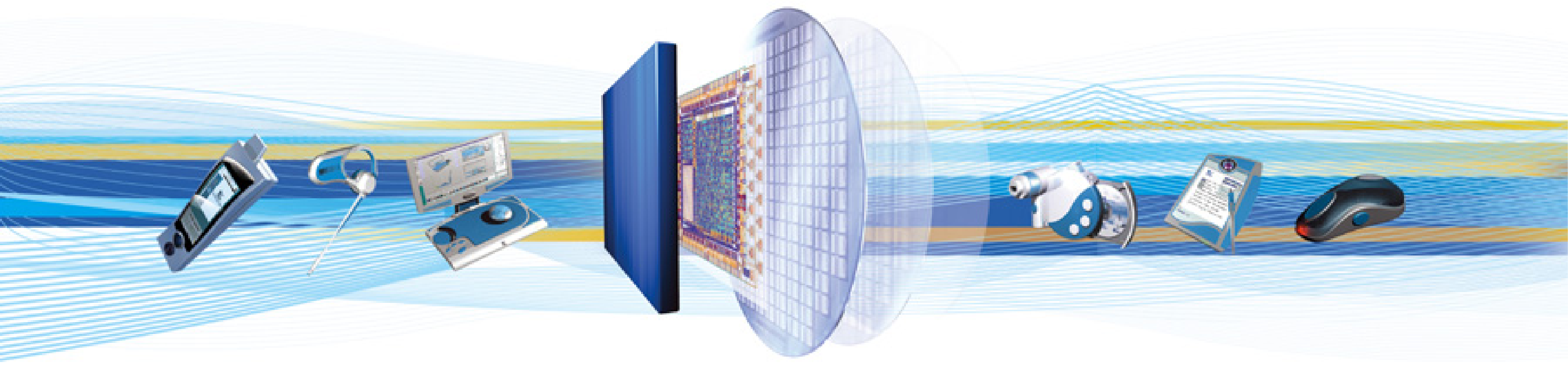
CSR Synergy®

CSR Synergy Framework 3.1.0

DSPM API

August 2011



Cambridge Silicon Radio Limited

Churchill House
Cambridge Business Park
Cowley Road
Cambridge CB4 0WZ
United Kingdom

Registered in England and Wales 3665875

Tel: +44 (0)1223 692000
Fax: +44 (0)1223 692001
www.csr.com

api-0034

Contents

Tables:

1 Introduction

This document describes how to use the DSP Manager (DSPM) API. The DSPM API allows an application to control how audio data sources and sinks on the BlueCore™ are connected to each other, and additionally allows processing of these audio data streams. While the main purpose of this functionality is to process and route audio data, it is possible to process and route any kind of data.

1.1 Basic Concepts

The DSPM uses a number of abstractions to handle the various building blocks that can be used for setting up the routing and processing. These include:

1) Source

2) Sink

3) Operator

4) Connection

A source is, as the name implies, a source of a data stream, while a sink is a consumer of a data stream. To establish a flowing stream, at least one source and one sink is required. A connection can be established between a source and a sink. Doing so will cause the data to flow in a stream from the source to the sink. Sinks and sources of this type are referred to as *external*. Before using an external source or a sink, it must be opened using the `CSR_DSPM_STREAM_CP_OPEN_REQ` message. After use, it must be closed using the `CSR_DSPM_STREAM_CP_CLOSE_REQ` message. Connections are established using the `CSR_DSPM_CONNECTION_CREATE_REQ` message and deestablished using the `CSR_DSPM_CONNECTION_DESTROY_REQ` message. It is, however, not necessary to explicitly destroy connections, as these are automatically deestablished if either of the source/sink is closed.

Furthermore, operators can be created that processes the data flowing on a stream in various ways.There are different types of operators. An operator may have zero or several sources and sinks associated with it. Sinks and sources of this type are referred to as *internal*. This allows the operator to be connected to external sources and sinks (or the internal sources and sinks of other operators). The behaviour of an operator is called its *capability*. Each capability has a unique identifier, referred to as *capability identifier*. The general meaning of a given capability will never change; however, new features may be added to a capability in an upwards-compatible manner. Operators are created using the `CSR_DSPM_OPERATOR_CREATE_REQ` message, and after use destroyed using the `CSR_DSPM_OPERATOR_DESTROY_REQ` message. Destroying an operator will cause any connections established to its associated sources and sinks to be automatically deestablished and these connections shall not be explicitly destroyed in this case.

Collectively, sources and sinks are also referred to as connection points, and in the API the type `CsrDspmCpid` is used for containing a *connection point identifier* that uniquely identifies a specific source or sink. Similarly, the type `CsrDspmOpid` is used for containing an *operator identifier*, and the type `CsrDspmCid` for containing a *connection identifier*. These identifiers are assigned by DSPM and passed to the application in the message confirming the creation of the corresponding entity. The exception is for internal sources and sinks that are implicitly created when an operator is created. To obtain the *connection point identifiers* associated with an operator, the `CSR_DSPM_CPID_OPERATOR_SOURCE` and `CSR_DSPM_CPID_OPERATOR_SINK` macros must be used. The first argument is the *operator identifier* and the second parameter is the connection point number, which is an integer between zero and N - 1, where N is the number of sources or sinks (respectively) associated with the operator (this depends on the specific operator).

1.2 Patching Interfaces

DSPM provides two optional interfaces for registering callback functions that are called when patch data is required. If these callback functions are not installed the BlueCore™ will continue running with the firmware that is present in the device at the time of manufacturing. Please refer to the header file csr_dspm_lower.h which explains how to implement and register the appropriate callback functions.

2 Interface Description

2.1 Activate

This message must be sent to the DSPM by each application prior to using the rest of the interface to register the application as a user of the services provided by the DSPM. The DSPM will initialise itself when at least one application requires the services, and perform any necessary activation of hardware subsystems related to the functionality provided by this interface. When an application no longer requires the services provided by the DSPM interface, a `CSR_DSPM_DEACTIVATE_REQ` must be sent by each application to inform this to the DSPM. The DSPM will deinitialise itself and any hardware subsystems as necessary when no more applications require the services provided. Each application is expected to close/destroy/remove any acquired resources using the appropriate interfaces, before deactivating, or the resources will continue to be occupied in the hardware until it is reset or power cycled, rendering these resources unavailable to all applications.

Note that if an application sends a request without activating first, the request will be dropped silently and a confirm message will not be sent, regardless of whether other applications have activated the DSPM.

To send the `CSR_DSPM_ACTIVATE_REQ` primitive, the `CsrDspmActivateReqSend()` function is used. The function uses the arguments described in Table 1.

Type	Argument	Description
`CsrSchedQid`	`qid`	Queue identifier of application.

Table 1: Arguments to `CsrDspmActivateReqSend` function

When DSPM has processed the Activate request, a `CSR_DSPM_ACTIVATE_CFM` primitive will be sent back to the application. The primitive members are described in Table 2.

Type	Member	Description
`CsrPrim`	`type`	Signal identity – always set to `CSR_DSPM_ACTIVATE_CFM`

Table 2: Members in a CSR_DSPM_ACTIVATE_CFM primitive

2.2 Deactivate

See the description of CSR_DSPM_ACTIVATE_REQ in section **Error! Reference source not found.Error! Reference source not found.**.

To send the `CSR_DSPM_DEACTIVATE_REQ` primitive, the `CsrDspmDeactivateReqSend()` function is used. The function uses the arguments described in Table 3.

Type	Argument	Description
`CsrSchedQid`	`qid`	Queue identifier of application.

Table 3: Arguments to `CsrDspmDeactivateReqSend` function

When DSPM has processed the Deactivate request, a `CSR_DSPM_DEACTIVATE_CFM` primitive will be sent back to the application. The primitive members are described in Table 4.

Type	Member	Description
`CsrPrim`	`type`	Signal identity – always set to `CSR_DSPM_DEACTIVATE_CFM`

Table 4: Members in a CSR_DSPM_DEACTIVATE_CFM primitive

If the communication with the hardware is lost for some reason, the DSPM will issue a `CSR_DSPM_DEACTIVATE_IND` to activated applications, effectively cancelling their activation. If `CSR_BLUECORE_ONOFF` is defined, this will happen if the controlling application performs a deactivation. The DSPM will be in a deactivated state, and all resources that have been acquired through the DSPM interface are closed/destroyed/removed automatically. Any outstanding requests (except CSR_DSPM_ACTIVATE_REQ) that were sent prior to this indication as well as subsequent requests will be dropped silently and no confirm messages will be sent. To resume normal operation, the application must activate again by sending a `CSR_DSPM_ACTIVATE_REQ` message. The primitive members are described in Table 6.

Type	Member	Description
`CsrPrim`	`type`	Signal identity – always set to `CSR_DSPM_DEACTIVATE_IND`

Table 5: Members in a CSR_DSPM_DEACTIVATE_IND primitive

2.3 Stream Cp Open

This message is sent to DSPM to open one or several external sources or sinks. The message contains a list of `CsrDspmCpInfo` structures. Each entry in the list describes one source or one sink to open.

Type	Argument	Description
`CsrDspmCpid`	`cpid`	This parameter must be set to either `CSR_DSPM_CPID_SOURCE` or `CSR_DSPM_CPID_SINK` to indicate whether the connection point to open is a source or a sink.
`CsrDspmCpType`	`cpType`	This parameter can be selected from the list of connection point types defined in csr_dspm_prim.h (starting with `CSR_DSPM_CP_TYPE_`), or a numeric value if the desired connection point type is not listed.
`CsrUint16`	`instance`	The instance of the connection point. See description below.
`CsrUint16`	`channel`	The channel of the given instance of the connection point. See description below.

Table 6: Contents of `CsrDspmCpInfo` structure

The interpretation of the instance and channel parameters depend on the stream type. For PCM, I2S, CODEC, FM, SPDIF and DM, the instance parameter is the instance number (starting at 0) and the channel parameter is the channel number or PCM slot code (starting at 0). For stereo devices, channel 0 is the left channel and channel 1 is the right channel. The PCM slot code is either 0 for automatic selection or 0x8N00 to force the selection of slot N, where N is between 0 and 3. Please note that forced slot selection is only available on some BlueCore™ devices. If the BlueCore™ has a stereo CODEC, the channel number 2 may be used to create a single sink that sends to both channels.

For FASTPIPE the instance parameter is the pipe number (specifying pipe 0, the credit pipe, is not valid and the behaviour is undefined), and the channel parameter is ignored.

For SCO the instance parameter is the SCO handle and the channel parameter is ignored.

Opening A2DP as a source is not supported. For A2DP sinks, the instance parameter is the ACL and the channel parameter is the CID of the connection. The A2DP connection must have been established before opening the sink. Please note that A2DP is not available on some BlueCore™ devices.

The connection points are opened in the order they appear in the supplied list. If any of the connection points in the list cannot be opened for some reason, then no attempt is made to open subsequent connection points in the list.

To send the `CSR_DSPM_STREAM_CP_OPEN_REQ` primitive, the `CsrDspmStreamCpOpenReqSend()` function is used. The function uses the arguments described in Table 7.

Type	Argument	Description
`CsrSchedQid`	`qid`	Queue identifier of application.
`CsrUint8`	`cpInfoCount`	The number of connection points to open, and the length of the `cpInfo` list.
`CsrDspmCpInfo`	`*cpInfo`	List of `CsrDspmCpInfo` structures, describing the connection points to open.

Table 7: Arguments to `CsrDspmStreamCpOpenReqSend` function

When DSPM has processed the Stream Cp Open request, a `CSR_DSPM_STREAM_CP_OPEN_CFM` primitive will be sent back to the application. The primitive members are described in Table 8.

Type	Member	Description
`CsrPrim`	`type`	Signal identity – always set to `CSR_DSPM_STREAM_CP_OPEN_CFM`
`CsrResult`	`result`	`CSR_RESULT_SUCCESS` – all of the connection points have successfully been opened. The `successes` member will be equal to the `cpInfoCount` member. `CSR_DSPM_RESULT_UNSUPPORTED` – the BlueCore™ does not support this command. `CSR_DSPM_RESULT_UNAVAILABLE` – the BlueCore™ was unable to open one of the connection points either because it is in use by another application or because the BlueCore™ was unable to claim the connection point for some other reason. The `successes` member specifies how many connection points were successfully opened, while the state of the remaining connection points will remain unchanged.
`CsrUint8`	`successes`	The number of connection points that were successfully opened. If all connection points were opened successfully, `successes` will be equal to `cpInfoCount` and `result` will be `CSR_RESULT_SUCCESS`. If `successes` is less than `cpInfoCount`, `result` will be different from `CSR_RESULT_SUCCESS`.
`CsrUint8`	`cpInfoCount`	The number of connection points that was requested opened, and the length of the `cpInfo` list.
`CsrDspmCpInfo`	`*cpInfo`	The list of `CsrDspmCpInfo` structures supplied in the request is returned in the confirm message. The `cpid` member of each entry in the list will be updated to contain the assigned *connection point identifier*. If `successes` is less than `cpInfoCount`, only the first `successes` entries will be updated, and the rest will remain unmodified.

Table 8: Members in a CSR_DSPM_STREAM_CP_OPEN_CFM primitive

2.4 Stream Cp Close

This message is sent to DSPM to close one or several external sources or sinks. The message contains a list of *connection point identifiers*. Each entry in the list describes one source or one sink to close.

The connection points are closed in the order they appear in the supplied list. Closing a source or a sink will deestablish any connections to it.

To send the `CSR_DSPM_STREAM_CP_CLOSE_REQ` primitive, the `CsrDspmStreamCpCloseReqSend()` function is used. The function uses the arguments described in Table 9.

Type	Argument	Description
`CsrSchedQid`	`qid`	Queue identifier of application.
`CsrUint8`	`cpidCount`	The number of connection points to close, and the length of the `cpid` list.
`CsrDspmCpid`	`*cpid`	List of the *connection point identifiers* of the connection points to close.

Table 9: Arguments to `CsrDspmStreamCpCloseReqSend` function

When DSPM has processed the Stream Source Close request, a `CSR_DSPM_STREAM_CP_CLOSE_CFM` primitive will be sent back to the application. The primitive members are described in Table 10.

Type	Member	Description
`CsrPrim`	`type`	Signal identity – always set to `CSR_DSPM_STREAM_CP_CLOSE_CFM`
`CsrUint8`	`cpidCount`	The number of connection points that was requested closed, and the length of the `cpid` list.
`CsrDspmCpid`	`cpid`	The list of *connection point identifiers* supplied in the request is returned in the confirm message unmodified.

Table 10: Members in a CSR_DSPM_STREAM_CP_CLOSE_CFM primitive

2.5 Stream Sco Enable

If the application needs to open the source and/or sink associated with a SCO connection, the application must enable SCO streams first by sending the `CSR_DSPM_STREAM_SCO_ENABLE_REQ` message with the enable parameter set to `TRUE`, before establishing the SCO connection. This is a shared setting, and it should not be set to `FALSE`, unless the application is certain that no other entities rely on this to be enabled. The setting persists until reset so it is only necessary to set it once before the first SCO connection is established. The reset state of this setting depends on the value of `PSKEY_ENABLE_SCO_STREAMS`, so it may not be necessary to send this request at all.

To send the `CSR_DSPM_STREAM_SCO_ENABLE_REQ` primitive, the `CsrDspmStreamScoEnableReqSend()` function is used. The function uses the arguments described in Table 11.

Type	Argument	Description
`CsrSchedQid`	`qid`	Queue identifier of application.
`CsrBool`	`enable`	`TRUE` to enable SCO streams, `FALSE` to disable.

Table 11: Arguments to `CsrDspmStreamSinkCloseReqSend` function

When DSPM has processed the Stream Sco Enable request, a `CSR_DSPM_STREAM_SCO_ENABLE_CFM` primitive will be sent back to the application. The primitive members are described in Table 12.

Type	Member	Description
`CsrPrim`	`type`	Signal identity – always set to `CSR_DSPM_STREAM_SCO_ENABLE_CFM`
`CsrResult`	`result`	`CSR_RESULT_SUCCESS` – SCO streams have successfully been enabled or disabled. `CSR_DSPM_RESULT_UNSUPPORTED` – the BlueCore™ does not support this command. `CSR_DSPM_RESULT_UNAVAILABLE` – the BlueCore™ was unable to enable or disable SCO streams for some other reason.

Table 12: Members in a CSR_DSPM_STREAM_SCO_ENABLE_CFM primitive

2.6 Stream Configure

This message is used for configuring a feature of one or several specified external sources or sinks. The message contains a list of `CsrDspmConfigInfo` structures. Each entry in the list describes one configuration parameter for a specified source or sink.

Type	Argument	Description
`CsrDspmCpid`	`cpid`	The *connection point identifier* of the source or sink to configure.
`CsrUint16`	`feature`	The feature to configure.
`CsrUint32`	`value`	The value to configure for the specified feature.

Table 13: Contents of `CsrDspmConfigInfo` structure

The configuration parameters are applied in the order they appear in the supplied list. If any of the configuration parameters in the list cannot be applied for some reason, then no attempt is made to apply subsequent configuration parameters in the list.

To send the `CSR_DSPM_STREAM_CONFIGURE_REQ` primitive, the `CsrDspmStreamConfigureReqSend()` function is used. The function uses the arguments described in Table 14.

Type	**Argument**	**Description**
`CsrSchedQid`	`qid`	Queue identifier of application.
`CsrUint8`	`configInfoCount`	The number of configuration parameters to apply, and the length of the `configInfo` list.
`CsrDspmConfigInfo`	`*configInfo`	List of `CsrDspmConfigInfo` structures, describing the configuration parameters to apply.

Table 14: Arguments to `CsrDspmStreamConfigureReqSend` function

When DSPM has processed the Stream Configure request, a `CSR_DSPM_STREAM_CONFIGURE_CFM` primitive will be sent back to the application. The primitive members are described in Table 15.

Type	**Member**	**Description**
`CsrPrim`	`type`	Signal identity – always set to CSR_DSPM_STREAM_CONFIGURE_CFM
`CsrResult`	`result`	`CSR_RESULT_SUCCESS` – all of the configuration parameters have successfully been applied. The `successes` member will be equal to the `configInfoCount` member. `CSR_DSPM_RESULT_UNSUPPORTED` – the BlueCore™ does not support this command. `CSR_DSPM_RESULT_UNAVAILABLE` – the BlueCore™ was unable to apply one of the configuration parameters for some other reason. The `successes` member specifies how many configuration parameters were successfully applied, and the remaining configuration parameters will not be applied.
`CsrUint8`	`successes`	The number of configuration parameters that were successfully applied. If all configuration parameters were applied successfully, `successes` will be equal to `configInfoCount` and `result` will be `CSR_RESULT_SUCCESS`. If `successes` is less than `configInfoCount`, `result` will be different from `CSR_RESULT_SUCCESS`.
`CsrUint8`	`configInfoCount`	The number of configuration parameters that was requested applied, and the length of the `configInfo` list.
`CsrDspmConfigInfo`	`*configInfo`	The list of `CsrDspmConfigInfo` structures supplied in the request is returned in the confirm message unmodified.

Table 15: Members in a CSR_DSPM_STREAM_CONFIGURE_CFM primitive

2.7 Stream Sync

This message is sent to request that one or several pairs of connection points be synchronised to prevent timing drifts. The message contains a list of `CsrDspmSyncInfo` structures. Each entry in the list describes one pair of connection points to synchronise.

Type	**Argument**	**Description**
`CsrDspmCpid`	`cpid1`	The *connection point identifier* of the first source or sink to synchronise.

`CsrDspmCpid`	`cpid2`	The *connection point identifier* of the second source or sink to synchronise. May be zero to indicate that synchronization of cpid1 is to be removed.

Table 16: Contents of `CsrDspmSyncInfo` structure

The combination of connection points that can be synchronised is restricted:

1) cpid1 and cpid2 can both be external sources.
2) cpid1 and cpid2 can both be external sinks.
3) cpid1 and cpid2 can both be internal sources on different operators.
4) cpid1 and cpid2 can both be internal sinks on different operators.

If either cpid1 or cpid2 are already synchronized with other connection points, then all the connection points become synchronised together.

As a special case cpid2 can be zero, meaning that any existing synchronisation on cpid1 is to be removed.

The synchronisation requests are applied in the order they appear in the supplied list. If any of the synchronisation requests in the list cannot be applied for some reason, then no attempt is made to apply subsequent synchronisation requests in the list.

To send the `CSR_DSPM_STREAM_SYNC_REQ` primitive, the `CsrDspmStreamSyncReqSend()` function is used. The function uses the arguments described in Table 17.

Type	**Argument**	**Description**
`CsrSchedQid`	`qid`	Queue identifier of application.
`CsrUint8`	`syncInfoCount`	The number of connection point pairs to synchronise, and the length of the `syncInfo` list.
`CsrDspmSyncInfo`	`*syncInfo`	List of `CsrDspmSyncInfo` structures, describing pairs of connection points to synchronise.

Table 17: Arguments to `CsrDspmStreamSyncReqSend` function

When DSPM has processed the Stream Sync request, a `CSR_DSPM_STREAM_SYNC_CFM` primitive will be sent back to the application. The primitive members are described in Table 18.

Type	**Member**	**Description**
`CsrPrim`	`type`	Signal identity – always set to CSR_DSPM_STREAM_SYNC_CFM
`CsrResult`	`result`	`CSR_RESULT_SUCCESS` – all of the synchronisation requests have successfully been applied. The `successes` member will be equal to the `syncInfoCount` member. `CSR_DSPM_RESULT_UNSUPPORTED` – the BlueCore™ does not support this command. `CSR_DSPM_RESULT_UNAVAILABLE` – the BlueCore™ was unable to apply one of the synchronisation requests for some other reason. The `successes` member specifies how many synchronisation requests were successfully applied, and the remaining synchronisation requests will not be applied.

Type	Member	Description
`CsrUint8`	`successes`	The number of synchronisation requests that were successfully applied. If all synchronisation requests were applied successfully, `successes` will be equal to `syncInfoCount` and `result` will be `CSR_RESULT_SUCCESS`. If `successes` is less than `syncInfoCount`, `result` will be different from `CSR_RESULT_SUCCESS`.
`CsrUint8`	`syncInfoCount`	The number of synchronisation requests that were requested applied, and the length of the `syncInfo` list.
`CsrDspmSyncInfo`	`*syncInfo`	The list of `CsrDspmSyncInfo` structures supplied in the request is returned in the confirm message unmodified.

Table 18: Members in a CSR_DSPM_STREAM_SYNC_CFM primitive

2.8 Operator Create

This message is sent to request the creation of one or several operators with specified, possibly different, capabilities. The capabilities can be selected from the list of defines in csr_dspm_prim.h or specified as a numeric value if the desired capability is not present in the list. The message contains a list of `CsrDspmOperatorInfo` structures. Each entry in the list describes one operator to create.

Type	Argument	Description
`CsrDspmOpid`	`opid`	The *operator identifier*. Must be set to CSR_DSPM_OPID_INVALID in the request.
`CsrDspmCapability`	`capability`	The desired capability of the operator.

Table 19: Contents of `CsrDspmOperatorInfo` structure

The operators are created in the order they appear in the supplied list. If any of the operators in the list cannot be created for some reason, then no attempt is made to create subsequent operators in the list.

To send the `CSR_DSPM_OPERATOR_CREATE_REQ` primitive, the `CsrDspmOperatorCreateReqSend()` function is used. The function uses the arguments described in Table 20.

Type	Argument	Description
`CsrSchedQid`	`qid`	Queue identifier of application.
`CsrUint8`	`operatorInfoCount`	The number of operators to create, and the length of the `operatorInfo` list.
`CsrDspmOperatorInfo`	`*operatorInfo`	List of `CsrDspmOperatorInfo` structures, describing the operators to create.

Table 20: Arguments to `CsrDspmOperatorCreateReqSend` function

When DSPM has processed the Operator Create request, a `CSR_DSPM_OPERATOR_CREATE_CFM` primitive will be sent back to the application. The primitive members are described in Table 21.

Type	Member	Description
`CsrPrim`	`type`	Signal identity – always set to `CSR_DSPM_OPERATOR_CREATE_CFM`

Type	Member	Description
CsrResult	result	CSR_RESULT_SUCCESS – all of the operators have successfully been created. The successes member will be equal to the operatorInfoCount member. CSR_DSPM_RESULT_UNSUPPORTED – the BlueCore™ does not support this command. CSR_DSPM_RESULT_UNAVAILABLE – the BlueCore™ was unable to create one of the operators for some other reason. The successes member specifies how many operators were successfully created, and the remaining operators will not be created.
CsrUint8	successes	The number of operators that were successfully created. If all operators were created successfully, successes will be equal to operatorInfoCount and result will be CSR_RESULT_SUCCESS. If successes is less than operatorInfoCount, result will be different from CSR_RESULT_SUCCESS.
CsrUint8	operatorInfoCount	The number of operators that were requested created, and the length of the operatorInfoInfo list.
CsrDspmSyncInfo	*operatorInfo	The list of CsrDspmOperatorInfo structures supplied in the request is returned in the confirm message. The opid member of each entry in the list will be updated to contain the assigned *operator identifier*. If successes is less than operatorInfoCount, only the first successes entries will be updated, and the rest will remain unmodified.

Table 21: Members in a CSR_DSPM_OPERATOR_CREATE_CFM primitive

2.9 Operator Destroy

This message is sent to request that one or several of previously created operators are destroyed. When an operator is no longer needed, it must be destroyed to relinquish the resources associated with it. When an operator is destroyed any connections to any of its connection points are automatically deestablished, and these connections shall not be explicitly destroyed.

The operators are destroyed in the order they appear in the supplied list. If any of the operators in the list cannot be destroyed for some reason, then no attempt is made to destroy subsequent operators in the list. An operator can only be destroyed if it is stopped. If an operator that is not stopped, the attempt to destroy it will fail.

To send the CSR_DSPM_OPERATOR_DESTROY_REQ primitive, the CsrDspmOperatorDestroyReqSend() function is used. The function uses the arguments described in Table 22.

Type	Argument	Description
CsrSchedQid	qid	Queue identifier of application.
CsrUint8	opidCount	The number of *operator identifiers* in the list.
CsrDspmOpid	*opid	A list of *operator identifiers* to destroy. The length of the list must match the opidCount argument.

Table 22: Arguments to CsrDspmOperatorCreateReqSend function

When DSPM has processed the Operator Destroy request, a CSR_DSPM_OPERATOR_DESTROY_CFM primitive will be sent back to the application. The primitive members are described in Table 23.

Type	Member	Description
CsrPrim	type	Signal identity – always set to CSR_DSPM_OPERATOR_DESTROY_CFM

Type	Member	Description
CsrResult	result	CSR_RESULT_SUCCESS – all of the operators have successfully been destroyed. The successes member will be equal to the opidCount member. CSR_DSPM_RESULT_UNSUPPORTED – the BlueCore™ does not support this command. CSR_DSPM_RESULT_UNAVAILABLE – the BlueCore™ was unable to destroy one of the operators for some other reason. The successes member specifies how many operators were successfully destroyed, while the state of the remaining operators will remain unchanged.
CsrUint8	successes	The number of operators that were successfully destroyed. If all operators were destroyed successfully, successes will be equal to opidCount and result will be CSR_RESULT_SUCCESS. If successes is less than opidCount, result will be different from CSR_RESULT_SUCCESS.
CsrUint8	opidCount	The number of *operator identifiers* in the list.
CsrDspmOpid	*opid	The list of *operator identifiers* supplied in the request is returned in the confirm message unmodified.

Table 23: Members in a CSR_DSPM_OPERATOR_DESTROY_CFM primitive

2.10 Operator Message

Some operators support communication with the application by means of operator messages. This message is used for delivering one or several operator messages to specified operators and receiving the corresponding responses. The message contains a list of CsrDspmOperatorMessageInfo structures. Each entry in the list describes one operator message and the operator to which it is to be delivered.

Type	Argument	Description
CsrDspmOpid	opid	The *operator identifier* of the operator to deliver the message to.
CsrUint16	messageLength	The length of the operator message (in 16bit words).
CsrUint16	*message	The message contents. The interpretation depends on the specific operator.

Table 24: Contents of CsrDspmOperatorMessageInfo structure

The operator messages are delivered in the order they appear in the supplied list. When a response is successfully received from the operator, it replaces the original operator message in the list. If any of the operator messages in the list cannot be delivered successfully for some reason, the original operator message remains in the list and no attempt is made to deliver subsequent operator messages in the list, and the remaining original operator messages remain unmodified in the list.

To send the CSR_DSPM_OPERATOR_MESSAGE_REQ primitive, the CsrDspmOperatorMessageReqSend() function is used. The function uses the arguments described in Table 25.

Type	Argument	Description
CsrSchedQid	qid	Queue identifier of application.
CsrUint8	operatorMessageInfoCount	The number of operator messages to deliver, and the length of the operatorMessageInfo list.
CsrUint16	*operatorMessageInfo	List of CsrDspmOperatorMessageInfo structures, describing the operator messages to deliver.

Table 25: Arguments to CsrDspmOperatorCreateReqSend function

When DSPM has processed the Operator Message request, a `CSR_DSPM_OPERATOR_MESSAGE_CFM` primitive will be sent back to the application. The primitive members are described in Table 26.

Type	Member	Description
`CsrPrim`	`type`	Signal identity – always set to `CSR_DSPM_OPERATOR_MESSAGE_CFM`
`CsrResult`	`result`	`CSR_RESULT_SUCCESS` – all of the operator messages have successfully been delivered. The `successes` member will be equal to the `operatorMessageInfoCount` member. `CSR_DSPM_RESULT_UNSUPPORTED` – the BlueCore™ does not support this command. `CSR_DSPM_RESULT_UNAVAILABLE` – the BlueCore™ was unable to deliver one of the operator messages for some other reason. The `successes` member specifies how many operator messages were successfully delivered, and the remaining operator messages will not be delivered.
`CsrUint8`	`successes`	The number of operator messages that were successfully delivered. If all operator messages were delivered successfully, `successes` will be equal to `operatorMessageInfoCount` and `result` will be `CSR_RESULT_SUCCESS`. If `successes` is less than `operatorMessageInfoCount`, `result` will be different from `CSR_RESULT_SUCCESS`.
`CsrUint8`	`operatorMessageInfoCount`	The number of operator messages that were requested delivered, and the length of the `operatorMessageInfo` list.
`CsrDspmSyncInfo`	`*operatorMessageInfo`	The list of `CsrDspmOperatorMessageInfo` structures supplied in the request is returned in the confirm message. The message of each entry in the list will be updated to contain the received response from the operator. If `successes` is less than `operatorInfoCount`, only the first `successes` entries will be updated, and the rest will remain unmodified.

Table 26: Members in a CSR_DSPM_OPERATOR_MESSAGE_CFM primitive

The operator may also, asynchronously, send an operator message to the application. In this case, the application that created the operator will receive the `CSR_DSPM_OPERATOR_MESSAGE_IND` primitive described in Table 27.

Type	Member	Description
`CsrPrim`	`type`	Signal identity – always set to `CSR_DSPM_OPERATOR_MESSAGE_IND`
`CsrDspmOpid`	`opid`	The *operator identifier* of the operator that sent the message.
`CsrUint16`	`messageLength`	The length of the message (in 16 bit words).
`CsrUint16`	`*message`	The message contents. The interpretation depends on the specific operator.

Table 27: Members in a CSR_DSPM_OPERATOR_MESSAGE_IND primitive

2.11 Operator Start

This message is sent to request that one or several of previously created operators are started.

The operators are started in the order they appear in the supplied list. If any of the operators in the list cannot be started for some reason, then no attempt is made to start subsequent operators in the list. An operator can successfully be started even if it is already started (will have no effect). Starting an operator that has nothing connected to it, or insufficient connections may fail, depending on the operator.

To send the `CSR_DSPM_OPERATOR_START_REQ` primitive, the `CsrDspmOperatorStartReqSend()` function is used. The function uses the arguments described in Table 28.

Type	**Argument**	**Description**
`CsrSchedQid`	`qid`	Queue identifier of application.
`CsrUint8`	`opidCount`	The number of *operator identifiers* in the list.
`CsrDspmOpid`	`*opid`	A list of *operator identifiers* to start. The length of the list must match the `opidCount` argument.

Table 28: Arguments to `CsrDspmOperatorStartReqSend` function

When DSPM has processed the Operator Start request, a `CSR_DSPM_OPERATOR_START_CFM` primitive will be sent back to the application. The primitive members are described in Table 29.

Type	**Member**	**Description**
`CsrPrim`	`type`	Signal identity – always set to `CSR_DSPM_OPERATOR_START_CFM`
`CsrResult`	`result`	`CSR_RESULT_SUCCESS` – all of the operators have successfully been started. The `successes` member will be equal to the `opidCount` member. `CSR_DSPM_RESULT_UNSUPPORTED` – the BlueCore™ does not support this command. `CSR_DSPM_RESULT_UNAVAILABLE` – the BlueCore™ was unable to start one of the operators for some other reason. The `successes` member specifies how many operators were successfully started, while the state of the remaining operators will remain unchanged.
`CsrUint8`	`successes`	The number of operators that were successfully started. If all operators were started successfully, `successes` will be equal to `opidCount` and `result` will be `CSR_RESULT_SUCCESS`. If `successes` is less than `opidCount`, `result` will be different from `CSR_RESULT_SUCCESS`.
`CsrUint8`	`opidCount`	The number of *operator identifiers* in the list.
`CsrDspmOpid`	`*opid`	The list of *operator identifiers* supplied in the request is returned in the confirm message unmodified.

Table 29: Members in a CSR_DSPM_OPERATOR_START_CFM primitive

2.12 Operator Stop

This message is sent to request that one or several of previously created operators are stopped.

The operators are stopped in the order they appear in the supplied list. If any of the operators in the list cannot be stopped for some reason, then no attempt is made to stop subsequent operators in the list. An operator can be successfully stopped even if it is already stopped or never has been started (will have no effect).

To send the `CSR_DSPM_OPERATOR_STOP_REQ` primitive, the `CsrDspmOperatorStopReqSend()` function is used. The function uses the arguments described in Table 30.

Type	Argument	Description
`CsrSchedQid`	`qid`	Queue identifier of application.
`CsrUint8`	`opidCount`	The number of *operator identifiers* in the list.
`CsrDspmOpid`	`*opid`	A list of *operator identifiers* to stop. The length of the list must match the `opidCount` argument.

Table 30: Arguments to `CsrDspmOperatorStopReqSend` function

When DSPM has processed the Operator Stop request, a `CSR_DSPM_OPERATOR_STOP_CFM` primitive will be sent back to the application. The primitive members are described in Table 31.

Type	Member	Description
`CsrPrim`	`type`	Signal identity – always set to `CSR_DSPM_OPERATOR_STOP_CFM`
`CsrResult`	`result`	`CSR_RESULT_SUCCESS` – all of the operators have successfully been stopped. The `successes` member will be equal to the `opidCount` member. `CSR_DSPM_RESULT_UNSUPPORTED` – the BlueCore™ does not support this command. `CSR_DSPM_RESULT_UNAVAILABLE` – the BlueCore™ was unable to stop one of the operators for some other reason. The `successes` member specifies how many operators were successfully stopped, while the state of the remaining operators will remain unchanged.
`CsrUint8`	`successes`	The number of operators that were successfully stopped. If all operators were stopped successfully, `successes` will be equal to `opidCount` and `result` will be `CSR_RESULT_SUCCESS`. If `successes` is less than `opidCount`, `result` will be different from `CSR_RESULT_SUCCESS`.
`CsrUint8`	`opidCount`	The number of *operator identifiers* in the list.
`CsrDspmOpid`	`*opid`	The list of *operator identifiers* supplied in the request is returned in the confirm message unmodified.

Table 31: Members in a CSR_DSPM_OPERATOR_STOP_CFM primitive

2.13 Operator Reset

This message is sent to request that one or several of previously created operators are reset.

The operators are reset in the order they appear in the supplied list. If any of the operators in the list cannot be reset for some reason, then no attempt is made to reset subsequent operators in the list. An operator can be reset regardles of whether it is currently started or stopped. The exact meaning of a reset depends on the operator, but will always leave the operator stopped.

To send the `CSR_DSPM_OPERATOR_RESET_REQ` primitive, the `CsrDspmOperatorResetReqSend()` function is used. The function uses the arguments described in Table 32.

Type	Argument	Description
`CsrSchedQid`	`qid`	Queue identifier of application.
`CsrUint8`	`opidCount`	The number of *operator identifiers* in the list.
`CsrDspmOpid`	`*opid`	A list of *operator identifiers* to reset. The length of the list must match the `opidCount` argument.

Table 32: Arguments to `CsrDspmOperatorResetReqSend` function

When DSPM has processed the Operator Reset request, a `CSR_DSPM_OPERATOR_RESET_CFM` primitive will be sent back to the application. The primitive members are described in Table 33.

Type	Member	Description
`CsrPrim`	`type`	Signal identity – always set to `CSR_DSPM_OPERATOR_RESET_CFM`
`CsrResult`	`result`	`CSR_RESULT_SUCCESS` – all of the operators have successfully been reset. The `successes` member will be equal to the `opidCount` member. `CSR_DSPM_RESULT_UNSUPPORTED` – the BlueCore™ does not support this command. `CSR_DSPM_RESULT_UNAVAILABLE` – the BlueCore™ was unable to reset one of the operators for some other reason. The `successes` member specifies how many operators were successfully reset, while the state of the remaining operators will remain unchanged.
`CsrUint8`	`successes`	The number of operators that were successfully reset. If all operators were reset successfully, `successes` will be equal to `opidCount` and `result` will be `CSR_RESULT_SUCCESS`. If `successes` is less than `opidCount`, `result` will be different from `CSR_RESULT_SUCCESS`.
`CsrUint8`	`opidCount`	The number of *operator identifiers* in the list.
`CsrDspmOpid`	`*opid`	The list of *operator identifiers* supplied in the request is returned in the confirm message unmodified.

Table 33: Members in a CSR_DSPM_OPERATOR_RESET_CFM primitive

2.14 Connection Create

This message requests that one or several connections be established between multiple sources and sinks. Each source or sink can only have at most one connection. If either of the source or sink already have a connection, the request will fail. If both of the source and the sink are external, data flow will begin as soon as the source begins to provide data, or immediately, if the source is already generating data. For internal sources and sinks, the data flow will not start until the corresponding operators are started using the `CSR_DSPM_OPERATOR_START_REQ` primitive. The direction of data flow is always from source to sink. The message contains a list of `CsrDspmConnectionInfo` structures. Each entry in the list describes one connection between one source and one sink.

Type	Argument	Description
`CsrDspmCid`	`cid`	The *connection identifier*. Must be set to CSR_DSPM_CID_INVALID in the request.
`CsrDspmCpid`	`cpidSource`	The *connection point identifier* of the source.
`CsrDspmCpid`	`cpidSink`	The *connection point identifier* of the sink.

Table 34: Contents of `CsrDspmConnectionInfo` structure

The connections are established in the order they appear in the supplied list. If any of the connections in the list cannot be established for some reason, then no attempt is made to establish subsequent connections in the list.

To send the `CSR_DSPM_CONNECTION_CREATE_REQ` primitive, the `CsrDspmConnectionCreateReqSend()` function is used. The function uses the arguments described in Table 35.

Type	Argument	Description
`CsrSchedQid`	`qid`	Queue identifier of application.
`CsrUint8`	`connectionInfoCount`	Number of connections to establish, and the length of the `connectionInfo` list.

Type	Argument	Description
CsrDspmConnectionInfo	*connectionInfo	List of CsrDspmConnectionInfo structures, describing the connection to establish.

Table 35: Arguments to **CsrDspmConnectionCreateReqSend** function

When DSPM has processed the Connection Create request, a CSR_DSPM_CONNECTION_CREATE_CFM primitive will be sent back to the application. The primitive members are described in Table 36.

Type	Member	Description
CsrPrim	type	Signal identity – always set to CSR_DSPM_CONNECTION_CREATE_CFM
CsrResult	result	CSR_RESULT_SUCCESS – all of the connections have successfully been established. The successes member will be equal to the connectionInfoCount member. CSR_DSPM_RESULT_UNSUPPORTED – the BlueCore™ does not support this command. CSR_DSPM_RESULT_UNAVAILABLE – the BlueCore™ was unable to establish one of the connections for some other reason. The successes member specifies how many connections were successfully established, and the remaining connections will not be established.
CsrUint8	successes	The number of connections that were successfully established. If all connections were established successfully, successes will be equal to connectionInfoCount and result will be CSR_RESULT_SUCCESS. If successes is less than connectionInfoCount, result will be different from CSR_RESULT_SUCCESS.
CsrUint8	cpInfoCount	The number of connections that was requested established, and the length of the connectionInfo list.
CsrDspmCpInfo	*cpInfo	The list of CsrDspmConnectionInfo structures supplied in the request is returned in the confirm message. The cid member of each entry in the list will be updated to contain the assigned *connection identifier*. If successes is less than connectionInfoCount, only the first successes entries will be updated, and the rest will remain unmodified.

Table 36: Members in a CSR_DSPM_CONNECTION_CREATE_CFM primitive

2.15 Connection Destroy

This primitive is used for deestablishing one or several previously established connections. The message contains a list of *connection identifiers*. Each entry in the list describes one connection to deestablish.

Destroying the connection does not destroy the source or the sink and any connected operators will remain in their started/stopped state.

Please note that destroying an operator or closing a source or a sink will automatically deestablish any connections to it, and in this case, this primitive shall not be used to deestablish these connections, as the associated *connection identifiers* may have been reused by other applications.

The connections are closed in the order they appear in the supplied list.

To send the CSR_DSPM_CONNECTION_DESTROY_REQ primitive, the CsrDspmConnectionDestroyReqSend() function is used. The function uses the arguments described in Table 37.

Type	Argument	Description
`CsrSchedQid`	`qid`	Queue identifier of application.
`CsrUint8`	`cidCount`	Number of connections to deestablish, and the length of the `cid` list.
`CsrDspmCid`	`*cid`	List of the *connection identifiers* of the connections to deestablish.

Table 37: Arguments to `CsrDspmConnectionDestroyReqSend` function

When DSPM has processed the Connection Destroy request, a `CSR_DSPM_CONNECTION_DESTROY_CFM` primitive will be sent back to the application. The primitive members are described in Table 38.

Type	Member	Description
`CsrPrim`	`type`	Signal identity – always set to `CSR_DSPM_CONNECTION_DESTROY_CFM`
`CsrUint8`	`cidCount`	The number of connections that was requested deestablished, and the length of the `cid` list.
`CsrDspmCid`	`*cid`	The list of *connection identifiers* supplied in the request is returned in the confirm message unmodified.

Table 38: Members in a CSR_DSPM_CONNECTION_DESTROY_CFM primitive

2.16 Capability Download

The capabilities that can be used by operators can be extended by downloading additional capabilities. This primitive allows downloading of a new capability. After downloading the capability operators can be created that use this capability using the `CSR_DSPM_OPERATOR_CREATE_REQ` primitive.

To send the `CSR_DSPM_CAPABILITY_DOWNLOAD_REQ` primitive, the `CsrDspmCapabilityDownloadReqSend()` function is used. The function uses the arguments described in Table 39.

Type	Argument	Description
`CsrSchedQid`	`qid`	Queue identifier of application.
`CsrDspmCapability`	`capability`	The capability to download.
`CsrUint16`	`versionMajor`	The major version of the capability.
`CsrUint16`	`versionMinor`	The minor version of the capability.
`CsrMblk`	`*data`	The capability code encapsulated in a `CsrMblk`.

Table 39: Arguments to `CsrDspmCapabilityDownloadReqSend` function

When DSPM has processed the Capability Download request, a `CSR_DSPM_CAPABILITY_DOWNLOAD_CFM` primitive will be sent back to the application. The primitive members are described in Table 40.

Type	Member	Description
`CsrPrim`	`type`	Signal identity – always set to `CSR_DSPM_CAPABILITY_DOWNLOAD_CFM`
`CsrResult`	`result`	`CSR_RESULT_SUCCESS` – the capability has successfully been downloaded. `CSR_DSPM_RESULT_UNSUPPORTED` – the BlueCore™ does not support this command, or the BlueCore™ does not support the supplied capability due to a minor/major version mismatch. `CSR_DSPM_RESULT_UNAVAILABLE` – the BlueCore™ was unable to download the capability for some other reason.
`CsrDspmCapability`	`capability`	The capability that was downloaded.

Table 40: Members in a CSR_DSPM_CAPABILITY_DOWNLOAD_CFM primitive

2.17 Capability Remove

A capability that was downloaded using the CSR_DSPM_CAPABILITY_DOWNLOAD_REQ primitive can be removed again by using this primitive. It is not possible to remove a capability that was not downloaded. Before removing a capability, any operators using this capability must first be destroyed or it will have no effect.

To send the `CSR_DSPM_CAPABILITY_REMOVE_REQ` primitive, the `CsrDspmCapabilityRemoveReqSend()` function is used. The function uses the arguments described in Table 41.

Type	Argument	Description
`CsrSchedQid`	`qid`	Queue identifier of application.
`CsrDspmCapability`	`capability`	The capability to remove.

Table 41: Arguments to `CsrDspmCapabilityRemoveReqSend` function

When DSPM has processed the Capability Remove request, a `CSR_DSPM_CAPABILITY_REMOVE_CFM` primitive will be sent back to the application. The primitive members are described in Table 42.

Type	Member	Description
`CsrPrim`	`type`	Signal identity – always set to `CSR_DSPM_CAPABILITY_REMOVE_CFM`
`CsrResult`	`result`	`CSR_RESULT_SUCCESS` – the capability has successfully been removed. `CSR_DSPM_RESULT_UNSUPPORTED` – the BlueCore™ does not support this command. `CSR_DSPM_RESULT_UNAVAILABLE` – the BlueCore™ was unable to remove the capability for some other reason.
`CsrDspmCapability`	`capability`	The capability that was removed.

Table 42: Members in a CSR_DSPM_CAPABILITY_REMOVE_CFM primitive

3 Document References

Document	**Reference**

Document History

Revision	Date	History
1	2010-02-24	Ready for release 3.0.2
2	2010-04-01	Ready for release 3.1.0

TradeMarks, Patents and Licences

Life Support Policy and Use in Safety-critical Compliance

Performance and Conformance

Refer to www.csrsupport.com for compliance and conformance to standards information.