Product Database

Component Design Document

1 Description

The product database component maintains a database of data product items. Only the latest single copy of each data product item is stored, and that value can be updated or fetched by ID via connectors. The component is configured by passing the minimum and maximum data product ID that the database can accept. The component allocates memory on the heap to store a maximum sized data product for every ID in range from the minimum to maximum ID provided. Invalid IDs received during requests are reported as events. The lookup algorithm is extremely fast, using the data product ID itself as a direct index into the database.
Note that IDs stored in this database should come from a compact ID space for most efficient memory usage. If you are manually setting the data product ID bases in your assembly model and creating a sparse ID set then this database component should not be used, as it could waste an enormous amount of memory. This component is designed to work best with the default, Adamant-allocated ID space for data products which spans from 1 to number of data products used in the system.

2 Requirements

The requirements for the Product Database component are specified below.

1. The component shall store a single, latest copy of each data product in the system.
2. The component shall update a data product upon request.
3. The component shall return a data product upon request.
4. The component shall reject update or return requests with unrecognized data product IDs.
5. The component shall provide a status upon data product return that indicates whether or not the data product is valid.
6. The component shall provide a command to dump any stored data product by ID.
7. The component shall provide a command to override the value of a stored data product with a commanded value.

3 Design

3.1 At a Glance

Below is a list of useful parameters and statistics that give a quick look into the makeup of the component.

- **Execution -** *passive*
- **Number of Connectors -** 8
- **Number of Invokee Connectors -** 3

- **Number of Invoker Connectors** - 5
- **Number of Generic Connectors** - *None*
- **Number of Generic Types** - *None*
- **Number of Unconstrained Arrayed Connectors** - *None*
- **Number of Commands** - 5
- **Number of Parameters** - *None*
- **Number of Events** - 18
- **Number of Faults** - *None*
- **Number of Data Products** - 2
- **Number of Data Dependencies** - *None*
- **Number of Packets** - 1

3.2 Diagram

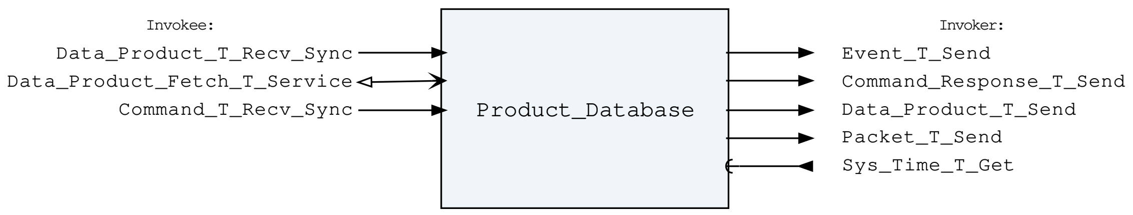

Figure 1: Product Database component diagram.

3.3 Connectors

Below are tables listing the component's connectors.

3.3.1 Invokee Connectors

The following is a list of the component's *invokee* connectors:

Table 1: Product Database Invokee Connectors

Name	Kind	Type	Return_Type	Count
`Data_Product_T_Recv_Sync`	`recv_sync`	`Data_Product.T`	-	1
`Data_Product_Fetch_T_Service`	`service`	`Data_Product_Fetch.T`	`Data_Product_Return.T`	1
`Command_T_Recv_Sync`	`recv_sync`	`Command.T`	-	1

Connector Descriptions:

- **`Data_Product_T_Recv_Sync`** - Store a data product item in the database.
- **`Data_Product_Fetch_T_Service`** - Fetch a data product item from the database.
- **`Command_T_Recv_Sync`** - This is the command receive connector. This does not need to be connected if the command for this component will not be used.

3.3.2 Invoker Connectors

The following is a list of the component's *invoker* connectors:

Table 2: Product Database Invoker Connectors

Name	Kind	Type	Return_Type	Count
`Event_T_Send`	send	`Event.T`	-	1
`Command_Response_T_Send`	send	`Command_Response.T`	-	1
`Data_Product_T_Send`	send	`Data_Product.T`	-	1
`Packet_T_Send`	send	`Packet.T`	-	1
`Sys_Time_T_Get`	get	-	`Sys_Time.T`	1

Connector Descriptions:

- **`Event_T_Send`** - Events are sent out of this connector.
- **`Command_Response_T_Send`** - This connector is used to register and respond to the component's commands. This does not need to be connected if the command for this component will not be used.
- **`Data_Product_T_Send`** - Data products are sent out of this connector.
- **`Packet_T_Send`** - Send a packet of data - used to dump database items.
- **`Sys_Time_T_Get`** - The system time is retrieved via this connector.

3.4 Interrupts

This component contains no interrupts.

3.5 Initialization

Below are details on how the component should be initialized in an assembly.

3.5.1 Component Instantiation

This component contains no instantiation parameters in its discriminant.

3.5.2 Component Base Initialization

This component contains no base class initialization, meaning there is no `init_Base` subprogram for this component.

3.5.3 Component Set ID Bases

This component contains commands, events, packets, faults, or data products that require a base identifier to be set at initialization. The `set_Id_Bases` procedure must be called with the following parameters:

Table 3: Product Database Set Id Bases Parameters

Name	Type
`Command_Id_Base`	`Command_Types.Command_Id_Base`
`Data_Product_Id_Base`	`Data_Product_Types.Data_Product_Id_Base`
`Event_Id_Base`	`Event_Types.Event_Id_Base`
`Packet_Id_Base`	`Packet_Types.Packet_Id_Base`

Parameter Descriptions:

- **Command_Id_Base** - The value at which the component's command identifiers begin.
- **Data_Product_Id_Base** - The value at which the component's data product identifiers begin.
- **Event_Id_Base** - The value at which the component's event identifiers begin.
- **Packet_Id_Base** - The value at which the component's unresolved packet identifiers begin.

3.5.4 Component Map Data Dependencies

This component contains no data dependencies.

3.5.5 Component Implementation Initialization

The calling of this implementation class initialization procedure is mandatory. This component requires the minimum and maximum acceptable data product IDs in order to size its internal database. Memory will be allocated to store a maximum sized data product for every ID in the range provided. The `init` subprogram requires the following parameters:

Table 4: Product Database Implementation Initialization Parameters

Name	Type	Default Value
`Minimum_Data_Product_Id`	`Data_Product_Types. Data_Product_Id`	*None provided*
`Maximum_Data_Product_Id`	`Data_Product_Types. Data_Product_Id`	*None provided*
`Send_Event_On_Missing`	`Boolean`	`True`

Parameter Descriptions:

- **Minimum_Data_Product_Id** - The minimum data product identifier that the database will accept.
- **Maximum_Data_Product_Id** - The maximum data product identifier that the database will accept. This value combined with the Minimum_Data_Product_Id are used to allocate a table on the heap. Ids stored in this database should come from a compact Id space for most efficient memory usage.
- **Send_Event_On_Missing** - By default the product database will send an event every time a data product is fetched that is missing. Sometimes this is expected behavior and the message is annoying. This flag allows that event to be disabled permanently on startup if needed.

3.6 Commands

These are the commands for the Product Database.

Table 5: Product Database Commands

Local ID	Command Name	Argument Type
0	`Clear_Override`	`Data_Product_Id.T`
1	`Clear_Override_For_All`	-
2	`Override`	`Data_Product.T`
3	`Dump`	`Data_Product_Id.T`

4	Dump_Poly_Type	Data_Product_Poly_Extract.T

Command Descriptions:

- **Clear_Override** - Clear the override condition for the data product of the provided ID.
- **Clear_Override_For_All** - Clear the override condition for all data products in the product store.
- **Override** - Override the value of a data product in the data product store. The value of this data product will be fixed to the commanded value, ignoring all other updates, until the override is cleared.
- **Dump** - Dump the data product of the provided ID in a packet.
- **Dump_Poly_Type** - Dump the data product of the provided ID into a poly type based on the provided offset and length.

3.7 Parameters

The Product Database component has no parameters.

3.8 Events

Below is a list of the events for the Product Database component.

Table 6: Product Database Events

Local ID	Event Name	Parameter Type
0	Data_Product_Update_Id_Out_Of_Range	Data_Product_Id.T
1	Data_Product_Fetch_Id_Out_Of_Range	Data_Product_Id.T
2	Data_Product_Fetch_Id_Not_Available	Data_Product_Id.T
3	Override_Cleared	Data_Product_Id.T
4	Override_Cleared_For_All	-
5	Data_Product_Overridden	Data_Product_Header.T
6	Data_Product_Override_Serialization_Failure	Data_Product_Header.T
7	Data_Product_Override_Id_Out_Of_Range	Data_Product_Id.T
8	Data_Product_Clear_Override_Id_Out_Of_Range	Data_Product_Id.T
9	Data_Product_Dump_Id_Not_Available	Data_Product_Id.T
10	Data_Product_Dump_Id_Out_Of_Range	Data_Product_Id.T
11	Data_Product_Dumped	Data_Product_Header.T
12	Dumping_Data_Product_Poly_Type	Data_Product_Poly_Extract.T
13	Dumped_Data_Product_Poly_Type	Data_Product_Poly_Event.T

14	Data_Product_Dump_Poly_Id_Not_Available	Data_Product_Id.T
15	Data_Product_Dump_Poly_Id_Out_Of_Range	Data_Product_Id.T
16	Data_Product_Poly_Type_Extraction_Failed	Data_Product_Header.T
17	Invalid_Command_Received	Invalid_Command_Info.T

Event Descriptions:

- **`Data_Product_Update_Id_Out_Of_Range`** - A data product update was received with an ID that was out of range.
- **`Data_Product_Fetch_Id_Out_Of_Range`** - A data product fetch was received with an ID that was out of range.
- **`Data_Product_Fetch_Id_Not_Available`** - A data product fetch was received with an ID that has not yet been stored in the database.
- **`Override_Cleared`** - Override condition cleared for the data product of the provided ID.
- **`Override_Cleared_For_All`** - Override condition cleared for all data products.
- **`Data_Product_Overridden`** - Data product overridden by command.
- **`Data_Product_Override_Serialization_Failure`** - Data product override could not be completed due to a serialization error.
- **`Data_Product_Override_Id_Out_Of_Range`** - A data product override command was received with an ID that was out of range.
- **`Data_Product_Clear_Override_Id_Out_Of_Range`** - A data product clear override command was received with an ID that was out of range.
- **`Data_Product_Dump_Id_Not_Available`** - A data product dump command was received with an ID that has not yet been stored in the database.
- **`Data_Product_Dump_Id_Out_Of_Range`** - A data product dump command was received with an ID that was out of range.
- **`Data_Product_Dumped`** - Data product dumped into a packet by command.
- **`Dumping_Data_Product_Poly_Type`** - Data product poly type dumped into a packet by command.
- **`Dumped_Data_Product_Poly_Type`** - Data product poly type dumped into a packet by command.
- **`Data_Product_Dump_Poly_Id_Not_Available`** - A data product dump poly command was received with an ID that has not yet been stored in the database.
- **`Data_Product_Dump_Poly_Id_Out_Of_Range`** - A data product dump poly command was received with an ID that was out of range.
- **`Data_Product_Poly_Type_Extraction_Failed`** - A data product dump poly command failed because the extraction could not succeed with the provided parameters.
- **`Invalid_Command_Received`** - A command was received with invalid parameters.

3.9 Data Products

Data products for the Product Database component.

Table 7: Product Database Data Products

Local ID	Data Product Name	Type
`0x0000 (0)`	`Data_Product_Poly_Type_Dump`	`Data_Product_Poly_Type.T`
`0x0001 (1)`	`Database_Override`	`Packed_Enable_Disable_Type.T`

Data Product Descriptions:

- **`Data_Product_Poly_Type_Dump`** - Data product poly type dumped into a data product by command.
- **`Database_Override`** - If set to Enabled then the database contains at least one data product that has been overridden by command.

3.10 Data Dependencies

The Product Database component has no data dependencies.

3.11 Packets

Packets for the Product Database.

Table 8: Product Database Packets

Local ID	Packet Name	Type
`0x0000 (0)`	`Dump_Packet`	`Data_Product.T`

Packet Descriptions:

- **`Dump_Packet`** - This packet contains dumped data products.

3.12 Faults

The Product Database component has no faults.

4 Unit Tests

The following section describes the unit test suites written to test the component.

4.1 *Tests* Test Suite

This is a unit test suite for the Product Database Component

Test Descriptions:

- **`Test_Nominal_Scenario`** - This unit test tests the updating and fetching of data products with no errors.
- **`Test_Nominal_Override`** - This unit test tests the overriding and fetching of data products with no errors.
- **`Test_Nominal_Dump`** - This unit test tests the dumping of a data product packet with no errors.
- **`Test_Nominal_Dump_Poly`** - This unit test tests the sending of a data product poly type.
- **`Test_Data_Not_Available`** - This unit test tests the fetching of data that has not yet been stored.

- **`Test_Id_Out_Of_Range`** - This test tries to update and fetch data that has a bad id.
- **`Test_Invalid_Command`** - This unit test exercises that an invalid command throws the appropriate event.

5 Appendix

5.1 Preamble

This component contains no preamble code.

5.2 Packed Types

The following section outlines any complex data types used in the component in alphabetical order. This includes packed records and packed arrays that might be used as connector types, command arguments, event parameters, etc..

Command.T:

Generic command packet for holding arbitrary commands

Table 9: Command Packed Record : 2080 bits *(maximum)*

Name	**Type**	**Range**	**Size (Bits)**	**Start Bit**	**End Bit**	**Variable Length**
`Header`	`Command_Header.T`	-	40	0	39	-
`Arg_Buffer`	`Command_Types.Command_Arg_Buffer_Type`	-	2040	40	2079	`Header.Arg_Buffer_Length`

Field Descriptions:

- **`Header`** - The command header
- **`Arg_Buffer`** - A buffer that contains the command arguments

Command_Header.T:

Generic command header for holding arbitrary commands

Table 10: Command_Header Packed Record : 40 bits

Name	**Type**	**Range**	**Size (Bits)**	**Start Bit**	**End Bit**
`Source_Id`	`Command_Types.Command_Source_Id`	0 to 65535	16	0	15
`Id`	`Command_Types.Command_Id`	0 to 65535	16	16	31
`Arg_Buffer_Length`	`Command_Types.Command_Arg_Buffer_Length_Type`	0 to 255	8	32	39

Field Descriptions:

- **`Source_Id`** - The source ID. An ID assigned to a command sending component.

- **Id** - The command identifier
- **Arg_Buffer_Length** - The number of bytes used in the command argument buffer

Command_Response.T:

Record for holding command response data.

Table 11: Command_Response Packed Record : 56 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Source_Id	Command_Types.Command_Source_Id	0 to 65535	16	0	15
Registration_Id	Command_Types.Command_Registration_Id	0 to 65535	16	16	31
Command_Id	Command_Types.Command_Id	0 to 65535	16	32	47
Status	Command_Enums.Command_Response_Status.E	0 => Success 1 => Failure 2 => Id_Error 3 => Validation_Error 4 => Length_Error 5 => Dropped 6 => Register 7 => Register_Source	8	48	55

Field Descriptions:

- **Source_Id** - The source ID. An ID assigned to a command sending component.
- **Registration_Id** - The registration ID. An ID assigned to each registered component at initialization.
- **Command_Id** - The command ID for the command response.
- **Status** - The command execution status.

Data_Product.T:

Generic data product packet for holding arbitrary data types

Table 12: Data_Product Packed Record : 344 bits *(maximum)*

Name	Type	Range	Size (Bits)	Start Bit	End Bit	Variable Length
Header	Data_Product_Header.T	-	88	0	87	-
Buffer	Data_Product_Types.Data_Product_Buffer_Type	-	256	88	343	Header.Buffer_Length

Field Descriptions:

- **Header** - The data product header
- **Buffer** - A buffer that contains the data product type

Data_Product_Fetch.T:

A packed record which holds information for a data product request.

Table 13: Data_Product_Fetch Packed Record : 16 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Id	Data_Product_Types. Data_Product_Id	0 to 65535	16	0	15

Field Descriptions:
- **Id** - The data product identifier

Data_Product_Header.T:

Generic data_product packet for holding arbitrary data_product types

Table 14: Data_Product_Header Packed Record : 88 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Time	Sys_Time.T	-	64	0	63
Id	Data_Product_Types. Data_Product_Id	0 to 65535	16	64	79
Buffer_Length	Data_Product_ Types.Data_Product_ Buffer_Length_Type	0 to 32	8	80	87

Field Descriptions:
- **Time** - The timestamp for the data product item.
- **Id** - The data product identifier
- **Buffer_Length** - The number of bytes used in the data product buffer

Data_Product_Id.T:

A packed record which holds a data product identifier.

Table 15: Data_Product_Id Packed Record : 16 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Id	Data_Product_Types. Data_Product_Id	0 to 65535	16	0	15

Field Descriptions:
- **Id** - The data product identifier

Data_Product_Poly_Event.T:

Data product with 4 byte data buffer.

Table 16: Data_Product_Poly_Event Packed Record : 120 bits

Name	**Type**	**Range**	**Size (Bits)**	**Start Bit**	**End Bit**
Header	Data_Product_Header.T	-	88	0	87
Data	Basic_Types.Poly_32_Type	-	32	88	119

Field Descriptions:

- **Header** - The data product header
- **Data** - The polymorphic type.

Data_Product_Poly_Extract.T:

Contains information to extract a poly type from a data product. *Preamble (inline Ada definitions):*

```
subtype Data_Product_Bit_Offset_Type is Natural range Natural'First ..
↪   Data_Product_Types.Data_Product_Buffer_Type'Length *
↪   Basic_Types.Byte'Object_Size; subtype Poly_Type_Size_Type is Positive range
↪   Positive'First .. 32;
```

Table 17: Data_Product_Poly_Extract Packed Record : 40 bits

Name	**Type**	**Range**	**Size (Bits)**	**Start Bit**	**End Bit**
Id	Data_Product_Types.Data_Product_Id	0 to 65535	16	0	15
Offset	Data_Product_Bit_Offset_Type	0 to 256	16	16	31
Size	Poly_Type_Size_Type	1 to 32	8	32	39

Field Descriptions:

- **Id** - ID of the data product.
- **Offset** - Offset of the data product item (in bits).
- **Size** - Size of the data product item (in bits).

Data_Product_Poly_Type.T:

Data product poly type, for dumping arbitrary data products.

Table 18: Data_Product_Poly_Type Packed Record : 112 bits

Name	**Type**	**Range**	**Size (Bits)**	**Start Bit**	**End Bit**
Time	Sys_Time.T	-	64	0	63
Id	Data_Product_Types.Data_Product_Id	0 to 65535	16	64	79
Data	Basic_Types.Poly_32_Type	-	32	80	111

Field Descriptions:

- **`Time`** - The timestamp for the data product item.
- **`Id`** - The data product identifier
- **`Data`** - The polymorphic type.

Data_Product_Return.T:

This record holds data returned from a data product fetch request.

Table 19: Data_Product_Return Packed Record : 352 bits *(maximum)*

Name	Type	Range	Size (Bits)	Start Bit	End Bit	Variable Length

`The_Status`	`Data_Product_Enums.Fetch_Status.E`	`0 => Success`
		`1 => Not_Available`
		`2 => Id_Out_Of_Range`
			8	0	7	-

`The_Data_Product`	`Data_Product.T`	-	344	8	351	-

Field Descriptions:

- **`The_Status`** - A status relating whether or not the data product fetch was successful or not.
- **`The_Data_Product`** - The data product item returned.

Event.T:

Generic event packet for holding arbitrary events

Table 20: Event Packed Record : 344 bits *(maximum)*

Name	Type	Range	Size (Bits)	Start Bit	End Bit	Variable Length
`Header`	`Event_Header.T`	-	88	0	87	-
`Param_Buffer`	`Event_Types.Parameter_Buffer_Type`	-	256	88	343	`Header.Param_Buffer_Length`

Field Descriptions:

- **`Header`** - The event header
- **`Param_Buffer`** - A buffer that contains the event parameters

Event_Header.T:

Generic event packet for holding arbitrary events

Table 21: Event_Header Packed Record : 88 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
`Time`	`Sys_Time.T`	-	64	0	63
`Id`	`Event_Types.Event_Id`	`0 to 65535`	16	64	79

Param_Buffer_Length	Event_Types. Parameter_Buffer_ Length_Type	0 to 32	8	80	87

Field Descriptions:

- **Time** - The timestamp for the event.
- **Id** - The event identifier
- **Param_Buffer_Length** - The number of bytes used in the param buffer

Invalid_Command_Info.T:

Record for holding information about an invalid command

Table 22: Invalid_Command_Info Packed Record : 112 bits

Name	**Type**	**Range**	**Size (Bits)**	**Start Bit**	**End Bit**
Id	Command_Types. Command_Id	0 to 65535	16	0	15
Errant_Field_ Number	Interfaces. Unsigned_32	0 to 4294967295	32	16	47
Errant_Field	Basic_Types.Poly_ Type	-	64	48	111

Field Descriptions:

- **Id** - The command Id received.
- **Errant_Field_Number** - The field that was invalid. 1 is the first field, 0 means unknown field, 2**32 means that the length field of the command was invalid.
- **Errant_Field** - A polymorphic type containing the bad field data, or length when Errant_Field_Number is 2**32.

Packed_Enable_Disable_Type.T:

Single component record for holding an enable/disable enumeration.

Table 23: Packed_Enable_Disable_Type Packed Record : 8 bits

Name	**Type**	**Range**	**Size (Bits)**	**Start Bit**	**End Bit**
State	Basic_Enums. Enable_Disable_ Type.E	0 => Disabled 1 => Enabled	8	0	7

Field Descriptions:

- **State** - The 8-bit enable disable enumeration.

Packet.T:

Generic packet for holding arbitrary data

Table 24: Packet Packed Record : 10080 bits *(maximum)*

Name	Type	Range	Size (Bits)	Start Bit	End Bit	Variable Length
Header	Packet_Header.T	-	112	0	111	-
Buffer	Packet_Types.Packet_Buffer_Type	-	9968	112	10079	Header.Buffer_Length

Field Descriptions:

- **Header** - The packet header
- **Buffer** - A buffer that contains the packet data

Packet_Header.T:

Generic packet header for holding arbitrary data

Table 25: Packet_Header Packed Record : 112 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Time	Sys_Time.T	-	64	0	63
Id	Packet_Types.Packet_Id	0 to 65535	16	64	79
Sequence_Count	Packet_Types.Sequence_Count_Mod_Type	0 to 16383	16	80	95
Buffer_Length	Packet_Types.Packet_Buffer_Length_Type	0 to 1246	16	96	111

Field Descriptions:

- **Time** - The timestamp for the packet item.
- **Id** - The packet identifier
- **Sequence_Count** - Packet Sequence Count
- **Buffer_Length** - The number of bytes used in the packet buffer

Sys_Time.T:

A record which holds a time stamp using GPS format including seconds and subseconds since epoch (1-5-1980 to 1-6-1980 midnight).

Table 26: Sys_Time Packed Record : 64 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Seconds	Interfaces.Unsigned_32	0 to 4294967295	32	0	31
Subseconds	Interfaces.Unsigned_32	0 to 4294967295	32	32	63

Field Descriptions:

- **Seconds** - The number of seconds elapsed since epoch.

- **Subseconds** - The number of 1/(2^32) sub-seconds.

5.3 Enumerations

The following section outlines any enumerations used in the component.

Basic_Enums.Enable_Disable_Type.E:

This enumeration includes enable and disable state.

Table 27: Enable_Disable_Type Literals:

Name	Value	Description
Disabled	0	The state is disabled.
Enabled	1	The state is enabled.

Command_Enums.Command_Response_Status.E:

This status enumeration provides information on the success/failure of a command through the command response connector.

Table 28: Command_Response_Status Literals:

Name	Value	Description
Success	0	Command was passed to the handler and successfully executed.
Failure	1	Command was passed to the handler not successfully executed.
Id_Error	2	Command id was not valid.
Validation_Error	3	Command parameters were not successfully validated.
Length_Error	4	Command length was not correct.
Dropped	5	Command overflowed a component queue and was dropped.
Register	6	This status is used to register a command with the command routing system.
Register_Source	7	This status is used to register command sender's source id with the command router for command response forwarding.

Data_Product_Enums.Fetch_Status.E:

This status denotes whether a data product fetch was successful.

Table 29: Fetch_Status Literals:

Name	Value	Description
Success	0	The data product was returned successfully.
Not_Available	1	No data product is yet available for the provided id.
Id_Out_Of_Range	2	The data product id was out of range.