LLSM INTERFACE DEFINITION

BIT LAGOON, INC.

2750 E Spring Street Suite 240, Long Beach CA 90806

Document Prepared for:

Alan Kliner

Lamlinks Corporation

560 North Larchmont Boulevard

Los Angeles, CA 90004

ARCHITECTURE

The LLSM System is a distributed computing service framework which relies on database tables to provide inter-process communication and system configuration information. The system is primarily composed of "Nodes", "Services" and "Instances"

Node – A computer or workstation running the Lamlinks Service Manager executable.

Service – A feature or function of the LLSM system which can be used to perform a task which may be administrative, batch or real-time.

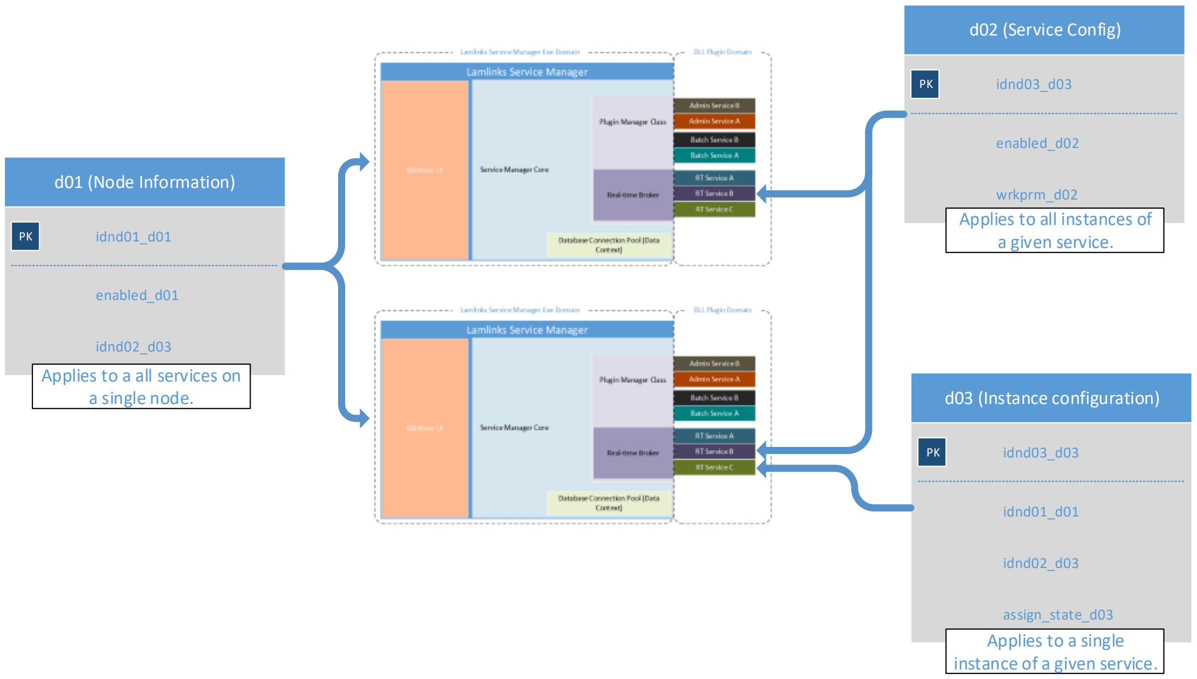
Instance – A service function hosted by a Lamlinks Service Manager executable node. Each service is defined in a self-contained dynamically loaded runtime library.

Nodes processes must be manually started or stopped via the windows UI and involves the starting and stopping of the Lamlinks Service Manager process. The table poem_db1.d01 contains a record for each node which has run the Lamlinks Service Manager executable. This table contains entries listing the machine identity as well as information regarding the last time the node was started, the last time the node updated its watchdog information and a field which specifies whether the node is enabled or not. A disabled node will run the Lamlinks Service Manager executable on the machine but all services will be stopped.

Services may be configured globally via the poem_db1.d02_tab service definition. The poem d02 definition mirrors that within the lis_db1 database. Each node scans the d02 table for changes at a predefined interval. If the enabled or wrkprm content related to a hosted instance has changed the node's plugin manager will start, stop or reload the service plugin depending on the current and desired states of the service instance.

Individual service instances (a service running on a specific machine node) may be configured via the poem_db1.d03_tab table. Similar to the d02 table, the d03 table mirrors the lis_db1 definition. In this table only the assign_state_d03 is used to serve as command information. The svc_status_d03 field provides a feedback mechanism to determine the current state of the instance. In the future the maxact_d03 and rankno_d03 may be implemented to help automate the assignment of node assign_state_d03 and to prevent the activation of more instances than desired by the service designer.

Organization of command and control database information:



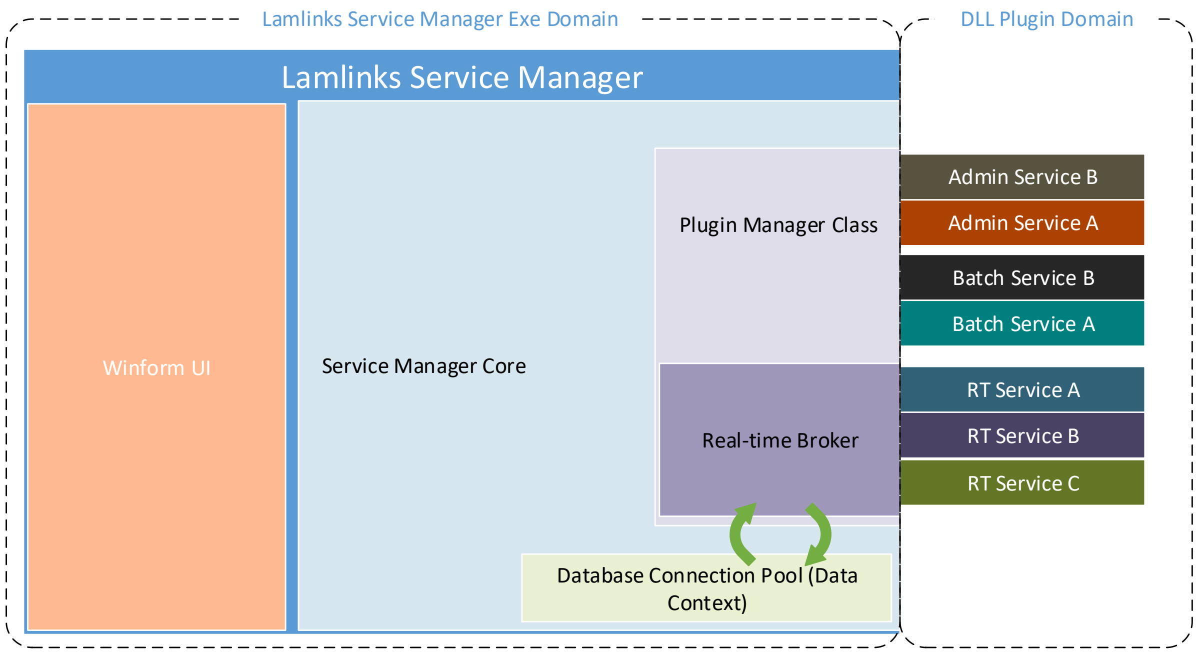
Administrative services provide internal support to the LLSM system by doing house-keeping or performing watchdog type activities. These functions are generally considered non-essential, as the instances of other service types should not typically rely on an administrative function to run.

Batch services are self-contained plugin programs that perform their task based on self-defined triggers, such as timers or database content changes. Most batch services are configured and controlled entirely through the service control parameters stored in d02. Once enabled the batch functions run without intervention or input from any other source.

Real-time services are plugins which respond to requests published through the poem_db1.j87_tab table. The real-time service interface is managed by a special broker object which interrogates the j87 table, identifies outstanding requests then delegates them to the appropriate real time instance running on the node.

note prior implementations of LLSM did not use a broker and each real-time instance used its own timer and queries to collect outstanding requests.

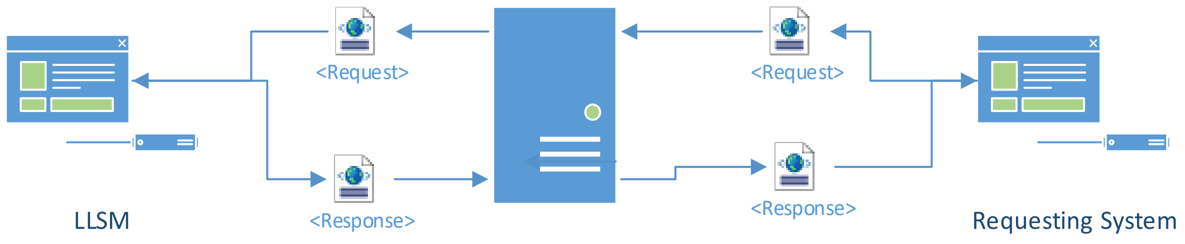
Basic organization of the software running on each node:



The real-time interface provide for bi-directional communication between requestor and responder through the use of xml formatted messages. The xml request and response formats follow a standard construction with naming and minimum field types set uniformly across all real-time functions. Fields specific to each function may be added to the xml format via extension, but the basic form shall remain in place.

The real-time request table (j87_tab) uses explicit fields to provide control and communication of the state of the request. All real-time requests intended to be serviced by the LLSM System shall be identified by reqsys_j87 = 'LLSM'. Requests to other systems may utilize the j87 table structure using an appropriate reqsys_j87 value. Each system may define the allowable set of values for control and communication fields such as rspcod_j87, reqfun_j87 and reqsta_j87. The LLSM specific implementations of these fields is described in the next section.

The following is a pictorial view of the real-time job queue architecture:



LLSM Real-Time Function Design

J87 Table Structure

The real-time interface consists primarily of records in the j87_tab table. In the context of the LLSM System this interface is referred to as the "Job Queue". Each Row in the j87_tab table is considered a "job".

Idnj87_j87 – auto numbered primary key. Used to identify / reference jobs across all aspects of the LLSM system including log entries.

Addtme_j87 – DateTime field representing the insertion time. This should be set by the requestor at insert.

Reqtme_j87 – DateTime field representing the time at or after which the LLSM system should process this request.

Reqkey_j87 – unique key generated by requesting system to identify the job request as needed. This field is not used or referenced with LLSM.

Toutby_j87 – DateTime after which the request is considered invalid. Jobs that have not completed by the time out are considered abandoned and should be cancelled.

Req_by_j87 – string identification of the requesting party. Typically a machine name.

Refun_j87 – string identification of the requested function. Defined by the function implementing the real-time interface.

Reqsta_j87 – request status. Within the context of the LLSM System the request status values are defined in the section reqsta_j87 (Request Status).

Reqxml_j87 – XML formatted request data. The format of this XML is described in the section reqxml_j87 (Request XML).

Rsptme_j87 – DateTime of the last response update. Set only by the responding LLSM node.

Rsp_by_j87 – string identification of the responding system node. Since only a single Lamlinks Service Manager executable may be run on a Machine at any given time, the string used is the Machine Name.

Rspcod_j87 – numeric response code. LLSM response codes are structurally uniform across all functions with a specific numeric range reserved for function-specific return values. See specific section for details.

Rspmsg_j87 – human readable text based message regarding the status of processing the job request.

Rspxml_j87 – XML formatted response data. The format is described in the section rspxml_j87 (Response XML)

Reqsys_j87 – string identifier which designates requests for a specific processing system. All LLSM jobs are identified by the reqsys_j87 'LLSM'.

REQSTA_J87 (REQUEST STATUS)

The following status values are defined by the LLSM System.

```
public const string REQUESTED = "requested";

public const string RESUBMITTED = "resubmitted";

public const string STARTING = "starting";

public const string INWORK = "processing";

public const string COMPLETE = "completed";

public const string CANCELLED = "cancelled";

public const string FUNCTION_UNAVAILABLE = "unavailable";

public const string INVALID = "invalid";
```

When initially submitted, the status should be set to requested. When the request is picked up by the broker on a node the state will be set to starting. Once the job has been delegated to a running service instance the status will be processing. If the LLSM system is unable to find an available instance of the requested service function the status will be set to unavailable.

A request which is being processed will exit to a status of completed if processing ran to completion. A request status of invalid is used to identify a job in which the request xml or other element of the request format is invalid. The invalid status indicates that the LLSM system was unable to understand the request. The job status of canceled is used to indicate that the request has not completed in the allotted time or that the requesting party no longer wishes to have the LLSM system complete the job. Jobs cancelled by the requestor are not guaranteed to be halted as this may not be practical in all cases.

REQXML_J87 (REQUEST XML)

The basic format for all requests is the same with the root tag being "Request" and the function specific variables declared within that root tag. The root tag must be named Request. This tag name is case sensitive. The xml encoding declaration tag is optional but recommended.

```
<?xml version="1.0" encoding="UTF-8"?><Request>VARIABLE DATA TAGS</Request>
```

Each function will declare the names and types of all expected variable tags. In the event a function does not require any request variables, an empty <Request></Request> tag is required. The requestor may provide additional tags not expressly required by the function declaration provided that they do not conflict with or confound the information required by the function. Any additional tags is considered to be informational only and will not be used by the LLSM System in the processing of the request, but may be used by the requesting system to store state information or other pertinent data.

RSPCOD_J87 (RESPONSE CODE)

The LLSM system utilizes a uniform set of response codes which cover a wide range of common exit conditions. Response code values 20 through 29 are reserved for function specific exit conditions. Return values 20-29 are defined by each function independently. Many functions do not require and function specific codes and instead rely on the set of uniform responses entirely.

```
public const int IN_WORK = 0;

public const int OK = 1;

public const int GENERAL_LOGON_FAILURE = 2;

public const int INVALID_USER_PASS = 6;
public const int UNKNOWN_USER = 7;


//Reserved Response codes for function specific responses.
public const int FUNCTION_SPECIFIC_1 = 20;
public const int FUNCTION_SPECIFIC_2 = 21;
public const int FUNCTION_SPECIFIC_3 = 22;
public const int FUNCTION_SPECIFIC_4 = 23;
public const int FUNCTION_SPECIFIC_5 = 24;
public const int FUNCTION_SPECIFIC_6 = 25;
public const int FUNCTION_SPECIFIC_7 = 26;
public const int FUNCTION_SPECIFIC_8 = 27;
public const int FUNCTION_SPECIFIC_9 = 28;
public const int FUNCTION_SPECIFIC_10 = 29;


//Abnormal program terminations
public const int BOT_FAILURE = 90;
public const int PROGRAM_ERROR = 99;


//Issues external to BOT Software
public const int EXTERNAL_SOFTWARE_ISSUE = 100;
public const int UNABLE_TO_COMPLETE_RESUBMIT = 101;
public const int NETWORK_OR_CONNECTION_ERROR = 102;
public const int DIBBS_STOP_MESSAGE = 103;
public const int CERTIFICATE_SECURITY_ERROR = 104;


//Job Completion or Startup Resource Issues
public const int NO_SERVICE_AVAILABLE = 200;
public const int TIMEOUT_WHILE_PROCESSING = 201;
public const int POST_CANCELLATION_COMPLETION = 202;


//Request Related Errors

public const int INVALID_REQUEST_XML_FORMAT = 900;
```

RSPXML_J87 (RESPONSE XML)

The basic format for all LLSM responses is the same with the root tag being "Response" and the function specific variables declared within that root tag. The root tag must be named Response. This tag name is case sensitive. Within the root tag of the Response there is a <debug_info> tag containing one or more <DebugMessage> tags. If there are no DebugMessage tags the debug_info tag will be omitted. Each DebugMessage contains a message and a creation tag. The DebugMessage information is included for the purposes of troubleshooting features of the system and should not be used programmatically to extract information regarding the outcome of an individual job request. The xml encoding declaration tag is optional but recommended.

Function specific return variables are specified by the individual function interface documentation and will occur within the root Response tag.

Example:

```
<?xml version="1.0" encoding="utf-8"?>
<Response xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <debug_info>
    <DebugMessage>
      <message>Start new Request</message>
      <creation>2016-12-19T15:39:12.5017726-08:00</creation>
    </DebugMessage>
    <DebugMessage>
      <message>job 1 starting</message>
      <creation>2016-12-19T15:39:12.5127818-08:00</creation>
    </DebugMessage>
  </debug_info>
</Response>
```

Logging and Data Collection

The LLSM System is designed with the ability for detailed, expressive logging. Logging information is stored in the j76_LogData table. This logging table is shared among several components within the over-all Lamlinks systems architecture. The system_name field is used to provide the highest order separation of log information. All log entries from LLSM System components store their data with a system_name value of ‘LLSM’.

Many of the fields are self-explanatory and are used consistently across all systems, however the following is the LLSM definition for each column.

Uid – unique identifier, auto numbered integer.

Created – DateTime that the log entry information was created within the LLSM component. This time may be different than the insert time for the log entry as log entry data is submitted to the database asynchronously and depending on application workloads or in conditions where database connectivity has been lost for some period of time, the log entry data may be queued and inserted sometime after the actual information was generated.

Createdby – integer representing the d01 identity of the node which inserted the log entry data.

Changed / changedby – null values not used for ordinary logging.

Loglevel – integer representing entry class or severity.

- 0 – Informational Entry, used during testing or debugging to gather detailed information about execution states and variable values.
- 1 – Normal Logging state, information deemed necessary to be collected under the normal day to day operation of the LLSM system. This data may be used for audit or statistical analysis and does not represent a condition which indicates an issue within the application, system or data.
- 2- Warning, a logging entry submitted as a warning indicates a potential issue or may be a leading indicator that an issue may be on the horizon. Warnings typically involved conditions which are handled by the code in a trusted and effective way, but should not be occurring under normal operating conditions.

- 3 – Error, error conditions are situations where action should be taken by the system operator to address the cause of the error state. Errors may be related to internal programming issues, unexpected data or variable values, external systems which are no longer responding or other condition for which some intervention is warranted.

Message – string based human readable short description of the entry or problem statement.

Class – The application class or service function from which the log entry is originating.

Key1 – tracking information, used to identify classes of log entries across multiple functions or requests. Ie. Authentication Errors or disk-access failures.

Key2 – tracking information which groups multiple log entries into a cohesive set in reference to a specific activity, such as the processing of a real-time job request. For real-time requests the key2 value will be the idnj87_j87 identity of the request being processed. This allows the viewing of the progress of a single request across all functions and methods which may be providing log data.

Ip_address – null, unused at this time in LLSM.

Hostname – String based identifier, system environment variable Machine ID.

Request_json – Full log entry text. Often machine readable data which may be xml or json encoded. Used for communicating detailed information about application state related to a log entry.

Method - String name / identity of the method or procedure from which the log originated. Used in combination with the class it can be used to quickly identify the source of a log entry despite messages which may be similar across different parts of the system.

REAL-TIME FUNCTION INTERFACE SPECIFICATIONS

ASSIST – ASSIST DOCUMENTATION DOWNLOAD

Function Identity – "ASSIST_HTTP_DOWNLOAD"

Request Variables

```
        public string document_id { get; set; }
```

Response Variables

```
        public List<UrlDownloadResult> results = new List<UrlDownloadResult>();
        public DocumentOverview Overview = new DocumentOverview();
```

```
where:

    public class DocumentOverview
    {
        public string title { get; set; }
        public string scope { get; set; }
        public string status { get; set; }
        public string fsc { get; set; }
        public string document_date { get; set; }
        public string next_review_date { get; set; }
        public string doc_category { get; set; }
        public string distribution_statement { get; set; }

    }

    public class UrlDownloadResult
    {
        public string url_folder_id { get; set; }
        public string requested_url { get; set; }
        public int result_code { get; set; }
        public string message { get; set; }
        public List<DebugMessage> debug_info = new List<DebugMessage>();
        public List<SavedPage> pages = new List<SavedPage>();
    }

    public class SavedPage
    {
        public string page_url { get; set; }
        public string filename { get; set; }
        public DateTime creation { get; set; }
    }
```

Custom Response Codes - Default

DDA – DIBBS Download Agent

Function Identity – `"DIBBS_HTTP_DOWNLOAD"`

Request Variables

```
    public string login_name {get;set;}
    public string login_password {get; set;}
    public List<string> url_list = new List<string>();
    public string paging_collection_id { get; set; }
```

Response Variables

```
    public List<UrlDownloadResult> results = new List<UrlDownloadResult>();
```

```
where:
```

```
    public class UrlDownloadResult
    {
        public int url_folder_id { get; set; }
        public string requested_url { get; set; }
        public int result_code { get; set; }
        public string message { get; set; }
        public List<DebugMessage> debug_info = new List<DebugMessage>();
        public List<SavedPage> pages = new List<SavedPage>();
    }

    public class SavedPage
    {
        public string page_url { get; set; }
        public string filename { get; set; }
        public DateTime creation { get; set; }
    }
```

Custom Response Codes

```
    public const int PAGING_COLLECTION_NOT_FOUND = 20;
```

DPU – DIBBS Password Update

Function Identity - `"UPDATE_DIBBS_PASSWORD"`

Request Variables

```
        public string login_name { get; set; }
        public string login_password { get; set; }
        public string new_password { get; set; }
```

Response Variables

```
        public string Message { get; set; }
```

Custom Response Codes

```
        public const int NEW_PASSWORD_BLANK = 20;
        public const int NEW_PASSWORD_REJECTED = 21;
```

DQUS – DIBBS Quote Upload Service

Function Identity - "UPLOAD_DIBBS_QUOTES"

Request Variables

```
        public string login_name {get;set;}
        public string login_password {get; set;}
        public string quote_folder { get; set; }
```

Response Variables

```
        public string Message { get; set; }
```

Custom Response Codes

```
        public const int NO_FILES = 20;
        public const int NO_ACCEPTED_QUOTES = 21;
        public const int MULTIPLE_FILES = 22;
```

ETT – Email to HTML Template

Function Identity - "EMAIL_TO_HTMLTEMPLATE"

Request Variables

```
        public string file_path { get; set; }
```

Response Variables

```
        public int template_uid { get; set; }
```

Custom Response Codes

```
        public const int NO_FILE = 20;
```

MTH – Mail Tracking Handler

Function Identity - "TRACK_EMAIL"

Request Variables –

```
        public MailMessage Message { get; set; }
```

```
where:

  public class MailMessage
    {
        public enum SendType { Direct, Distribution, Tracked }
        public SendType SendVia = SendType.Direct;
        public enum SendFrom { Erin, Henry, Quotes }
        public SendFrom From = SendFrom.Henry;
        public string Subject { get; set; }
        public List<AttachmentDef> Attachments = new List<AttachmentDef>();
        public string HTMLBody { get; set; }
        public ContactList To = new ContactList();
        public ContactList Cc = new ContactList();
        public ContactList Bcc = new ContactList();
        public DateTime SendAfter = DateTime.Now;
        public string ReferenceKey { get; set; }
        public int? marketing_campaign_id { get; set; }
    }

    public class AttachmentDef
    {
        public string Path { get; set; }
        public string DisplayName { get; set; }
    }

    public class Contact
    {
        public string email_address { get; set; }
        public string name { get; set; }
    }
```

Response Format

```
        public List<int> sent_messages = new List<int>();
```

Custom Response Codes

```
        public const int TEMPLATE_NOT_FOUND = 20;
        public const int NO_TEMPLATE_OR_MESSAGE_BODY_SPECIFIED = 21;
```

P2T – PDF TO TEXT

Function Identity - `"PDF_TO_TEXT"`

Request Variables

```
        public string file_path { get; set; }
```

Response Format

```
        public string text { get; set; }
```

Custom Response Codes

```
        public const int NO_FILE = 20;
```

ACCURATE INFORMATION

All information provided by Bit Lagoon is considered factual and correct to the best of our knowledge at the time of writing. We cannot be held liable for inaccuracies arising from changes in 3rd party information or from the evolving nature of technology and knowledge. It is our mission to provide accurate and timely information and to never mislead; that is our pledge.

PRIVACY STATEMENT

Any and all intellectual property or proprietary information provided during the course of contracting with Bit Lagoon will remain confidential and shall not be disclosed to any 3rd parties without prior written consent. This intellectual property protection shall remain in effect for as long as the information remains within Bit Lagoon servers or records, or until the property owner has released the information publicly or provided express written consent to Bit Lagoon to release the information to a 3rd party.

DISCLAIMER

All software, designs, specification and other products provided by Bit Lagoon are provided as-is. Bit Lagoon makes not guarantee for fitness of use. It is your responsibility to carefully review all specifications to ensure that they will meet your end requirements. While Bit Lagoon makes every effort to develop specifications tailored to meet the needs of each customer, no guarantee can be made that all facets of the customer requirements are embodied in the final specification. Any product provided by Bit Lagoon in accordance with a Specification is developed and tested to conform to the specification. Once installed in the customer environment there are factors beyond the control of Bit Lagoon which may affect product function. Bit Lagoon is not liable for any damages arising from the installation or use of Bit Lagoon products or any products developed in accordance with a Bit Lagoon Product Specification.