# ChainBuilder ESB Architecture White Paper

Bostech Corporation

April 2008

# Table of Contents

## About This Document

This white paper provides a technical overview of Bostech ChainBuilder® ESB, a comprehensive Service Oriented Architecture development and runtime environment. The document describes the architecture overview of the product, the key features that ChainBuilder ESB provides to develop, integrate and manage services integration within and across an enterprise.

The audience for this document includes potential customers evaluating the ChainBuilder ESB product for their application integration, data integration or Service Oriented Architecture (SOA) development.

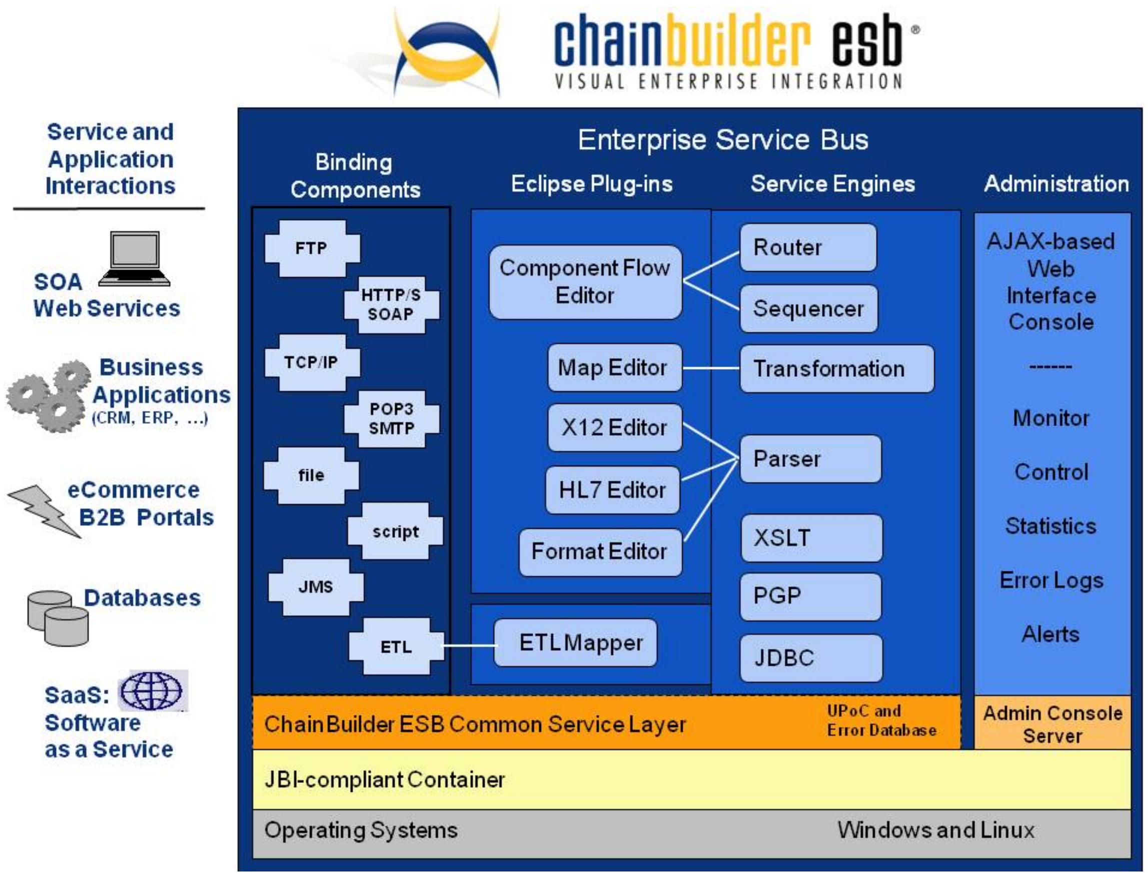
## Introduction

Information access and control remains a major hurdle for many organizations. Since markets continue to change rapidly, businesses need to react quickly to the changes. Information Technology (IT) needs to have an agile, service-oriented infrastructure to support rapid changes in the business to serve the business better. Many IT organizations adopt SOA principles and patterns as the best practice to build their IT infrastructure. The keys to SOA are reuse of services, agility and flexibility.

An Enterprise Service Bus (ESB) is known as the most economical and efficient solution to build out the service and integration layers in an SOA environment. An ESB provides the basic mediation capability such as service orchestration, routing, transformation and protocol brokering as well as additional services such as security, logging, alerts and auditing.

ChainBuilder ESB is an enterprise service bus targeted at rapid SOA development. ChainBuilder ESB consists of simple, easy-to-use GUI tools with drag-and-drop and wizard approach for designing a process flow or a map. ChainBuilder ESB focuses on application and data integration with existing IT assets such as legacy systems and databases, and supports open standards, industry-specific communication, and message standards.

The following diagram is the ChainBuilder ESB product schematic.



## Open Standard Based

ChainBuilder ESB is compliant with the single industry standard for building an Enterprise Service Bus – Java Business Integration (JBI). With that, developers can be assured their critical SOA IT asset is not locked in with one vendor.  Solution Architects have the choice to mix and match components and technologies from a list of JBI vendors and communities and the advantage of developing solutions with a standard application programming interface (API).

In addition, ChainBuilder ESB supports technology standards such as WSDL, HTTP, SOAP, JDBC, JMS, SOAP with Attachment, SSL as well as industry standards like Health Level 7 (HL7) and EDI.

## Composite Application Development

Instead of traditional application development where users either develop the application from the ground-up or reuse the existing IT asset at the library or framework level, ChainBuilder ESB enables SOA development by allowing users to assemble a composite application from pre-built JBI components which are supplied by a vendor or built in-house. Composite application development elevates code reuse to a new level.

## Integration with Disparate Systems

Leverage your investment in mature applications and databases with straight-forward integration paths to disparate systems. In addition to standard XML and Web Services support, you can use ChainBuilder ESB to create message definitions for fixed, variable, and other non-XML structure message formats and easily connect to non-Web Services protocols like FTP, File and TCP/IP.

## Easy to Develop

Bostech chose the popular Eclipse IDE as the standard development platform for ChainBuilder ESB. Developers can use this familiar interface with accompanying wizards and drag and drop functionality to easily create a process flow, map or other artifacts without hand coding.

## Easy to Extend

ChainBuilder ESB contains an extension framework called "User Point of Control (UPoC)" which allows users to write Java and Groovy code to be plugged into the bus. Bostech recognizes that integration is a complex task. Users are faced with unique integration requirements in their environment. The ability to extend the pre-built components is a must for a successful integration solution.

## Monitoring and Control

Operators can remotely manage the Enterprise Service Bus through ChainBuilder ESB's Administrative Console. Use this AJAX-based web interface to perform remote monitoring, administration and configuration of alerts on run-time JBI components and ChainBuilder ESB Service Assemblies.

## Architecture Overview

ChainBuilder ESB uses the Java Business Integration (JBI) standard that sets up a layered component based architecture. All integration services are provided by individual components. The architecture enables message content based, metadata-driven integration in all components with consistent exception and error handling and user point of control framework. All components have statistics support, logging, persistence of service endpoint settings and endpoint control.

### Java Business Integration

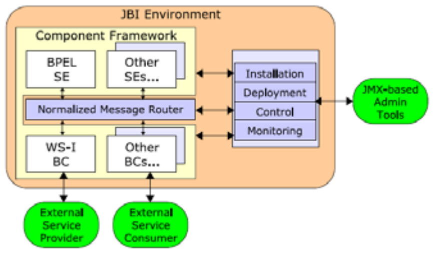
Java Business Integration (JBI) is a key differentiator of ChainBuilder ESB.  JBI serves as the cornerstone of the enterprise service bus.

JBI (JSR 208) is a standard specification created by Sun Microsystems and over twenty other major middleware infrastructure vendors.

The major characteristics of JBI are:

- Support within a single JVM or multiple JVMs with vendor implementation
- Plug-in components interoperate through the method of mediated message exchange. Supported message exchange patterns include InOnly, ReliableIn, InOut and InOptionalOut
- Exchange model based on WSDL
- Exchange infrastructure is provided by a NMR (Normalized Message Router)
- Contains Components of two distinct types
    - o **Service Engine (SE)** : providing business logic, transformation, … services as well as consuming of such services
    - o **Binding Component (BC)** : providing connectivity to services external to a JBI environment : communication protocols, services provided by EIS (Enterprise Information Systems), resources using remote access technology unavailable directly in Java
- Management structure based on JMX (Java Management eXtensions)

The following is the JBI architecture diagram:



## Layered Architecture

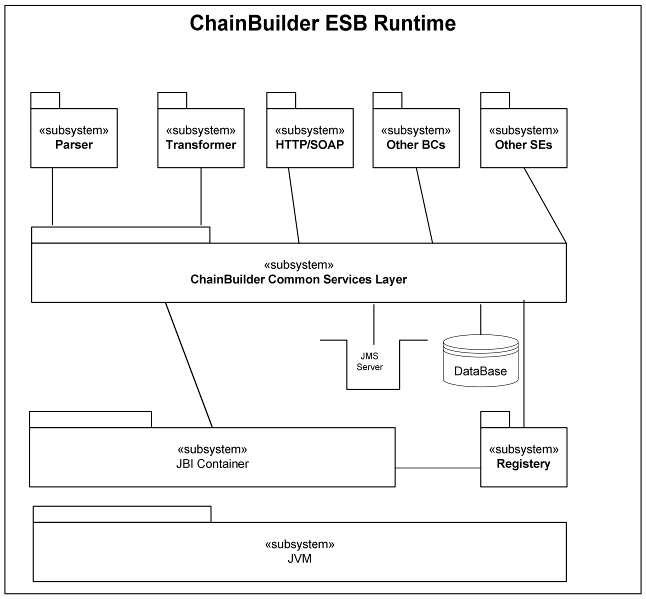
The ChainBuilder ESB runtime uses a layered architecture (see the diagram below).

At the bottom of the stack is the Java runtime environment (JVM). It supports both J2SE and J2EE Java environments. The next level up is the JBI container layer. ChainBuilder ESB is designed to be container-agnostic, although the Apache Servicemix container is currently included in the platform. Next to the JBI container is the Registry. In the current implementation, the Registry is internal to the JBI container. In the future, A UDDI-compliant Registry will be supported for service registration, versioning and service lookup.

The next level up in the stack is the ChainBuilder Common Services Layer (CCSL). The CCSL is Bostech's extension to the standard JBI container. The CCSL defines and implements many services that are critical to the ChainBuilder ESB product but are not defined in the JBI 1.0 spec. The examples of such services include User Points of Control (UPoC) via scripting, exception handling, transactional support, logging, auditing, etc.

ChainBuilder ESB includes JMS-compliant ActiveMQ in the stack to support reliable delivery, message queuing, store and forward, pub and sub and clustering. Also included in the architecture is a relational database (RDBMS) for the CCSL. Apache Derby is embedded into ChainBuilder ESB, but any RDBMS that supports JDBC can be used. Three distinct databases – *runtime*, *error* and *user* are defined in the ChainBuilder ESB architecture.

At the top of the ChainBuilder ESB stack, there are JBI-compliant components. The components provide the services necessary for an enterprise service bus.



## ChainBuilder Common Services Layer (CCSL)

The ChainBuilder Common Services Layer (CCSL) is a module that plugs in between JBI components and the JBI container. It provides a set of general services that can be useful for any component.

The CCSL is an optional layer. Since the CCSL is separate from and invisible to both the component and the container, the CCSL can work with any JBI components and containers. Both ChainBuilder ESB components and third-party JBI components can be configured to use the CCSL layer.

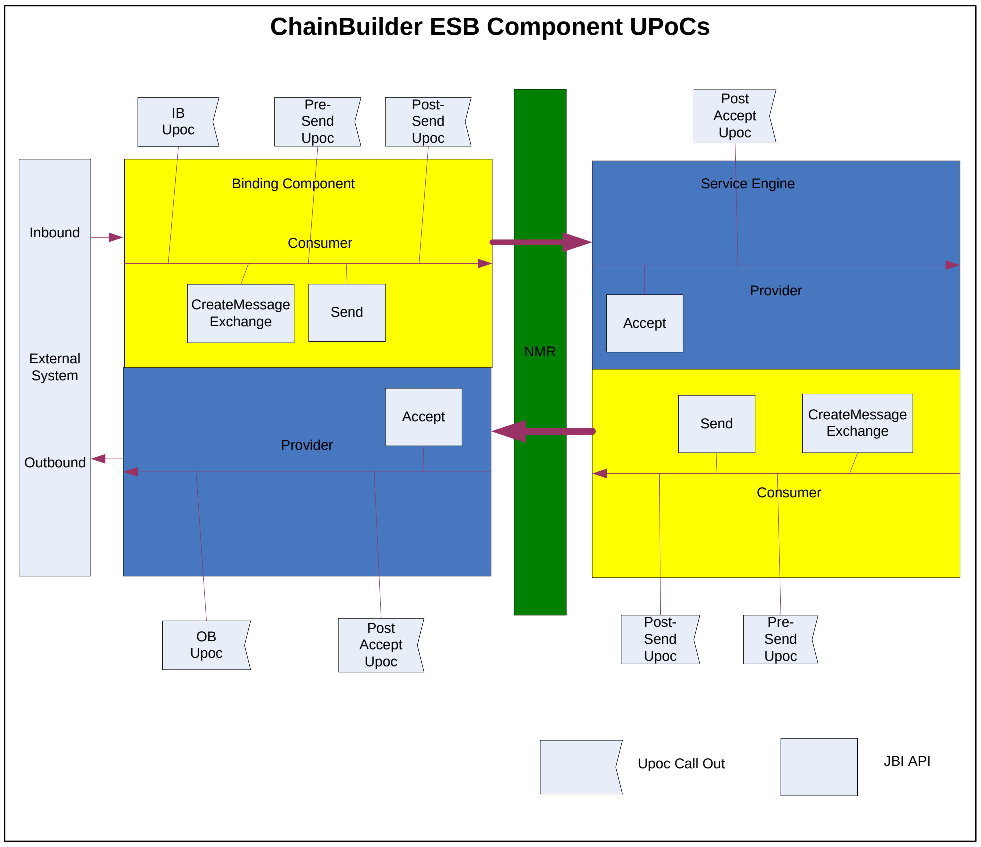
The CCSL currently provides the following functions.

- Allows for user scripting at various points of the exchange flow.
- Provides an exception handling mechanism that saves exchanges to a database in the event of processing exceptions.
- Provides conversion of the message content between raw XML and an enveloped, multi-record format used by other ChainBuilder ESB components.
- Provides standard implementation for scheduling, persistence of endpoint settings, statistics gathering and changes of logging.
- Provide an extension MBean for implementing endpoint control and component custom status.

## User Point of Control (UPoC) Framework

One of the most important and powerful features in ChainBuilder ESB is the *User Point of Control (UPoC)* framework. The UPoC framework provides a user-extensible method to control or change how a message gets processed in the engine at runtime. Java or any JSR-223 compatible scripting language can be used for scripting in ChainBuilder ESB. Groovy is the supported JRS-223 scripting language out-of-the-box.

The following diagram shows the 8 call out areas where UPoC scripts can be added:



As illustrated in above diagram, ChainBuilder ESB supports the following:

- **Inbound and outbound**:  Most of the transport protocol components in ChainBuilder ESB support a handler framework to allow users to write UPoC to customize how data is read from the transport connection before being turned into an ESB Normalized Message  and how a message is written to the transport connection.
- **Presend**: The UpoC is run immediately before a JBI DeliveryChannel's send() or sendSync() call. A presend UPoC may change the content or metadata of a message, the endpoint to get a message routed dynamically, or prevent a message from being sent.
- **Postsend**: The UPoC is run immediately after a call to sendSync() returns. When sendSync() returns, the out message is available in the exchange so this script allows you to process the response.
- **Postaccept**: The UPoC is run immediately after an exchange is received from a call to accept().
- **Start**: The UPoC is run immediately after the service unit starts. The exchange is null for the start context.
- **Stop**:  The UPoC is run immediately before the service unit stops. The exchange is null for the stop context.

## Exception Handling and the Error Database

ChainBuilder ESB provides consistent and robust exception handling for the services defined in the process flow. The Java Business Integration standard defines the Fault message in the message exchange to report an exception. The ChainBuilder ESB components or services catch the low-level Java exception and wrap it in a JBI fault, propagate it back to the high-level services and back to the service consumer.

ChainBuilder ESB triggers the standard error handling for the following:

- A low-level Java exception occurs in the component’s service provider logic
- A low-level Java exception or user-defined Java exception occurs in UPoC code or custom code in Script component
- A low-level or user-defined exception occurs in the user operation in message transformation
- A low-level or user-defined exception occurs in the custom code in routing services
- A message does not match any route defined in context-based routing
- A JBI exception occurs in exchanging a message between components and Normalized Message Router (NMR)

ChainBuilder ESB introduces two types of exceptions – the connection-related exception and the content-related exception. The error handling works in conjunction with JMS reliable delivery to retry any connection-related exception to minimize human intervention when an exception occurs. For content-related exceptions, the original request message along with exception cause are stored in the error database. The messages stored in the error database can be resent to the ChainBuilder ESB server after resolving the problem.

The ChainBuilder ESB Admin Console can be used to view the error database and diagnose errors. An alert can be triggered to notify users in case of error.

## Message Content and Metadata

The Java Business Integration standard defines message content and metadata in the form of a Message Exchange. The JBI component communicates with the Normalized Message Router (NMR) via a Message Exchange.

The content type in a Message Exchange supported by ChainBuilder ESB can be the following:

- XML, SOAP, or SOAP with Attachment
- Non-XML text when the MDL type of interface is expected
- Binary data in base64 encoded format for passing through

ChainBuilder ESB provides extensive metadata support in all its components. When a message exchange is received from the service consumer, metadata is attached to the message exchange as it passes through the process flow. The metadata and message content along with context information are made available to User Point of Control code to allow metadata-driven integration. The metadata in a message always overrides the default endpoint settings.

## Supported Messaging Patterns

An enterprise service bus needs to be able to communicate with applications or services in heterogeneous environments, adopting the messaging patterns of the external applications and service.  ChainBuilder ESB supports the following messaging patterns:

- **Synchronous request and response**: This is the most common messaging pattern supported by ChainBuilder ESB. It maps into the InOut Message Exchange Pattern in JBI. The consumer sends a request to the ESB and it blocks. The ESB processes the request and sends the response back.

- **Asynchronous request and response**: The asynchronous request and response messaging pattern is more reliable and tends to scale better compared to synchronous request and response. For example, when the back-end service requires human interaction, asynchronous request and response should be used. ChainBuilder ESB provides the mechanism to correlate the request and response via message ID and correlation ID.

- **Fire and forget**: The fire and forget pattern maps to the JBI’s ReliableIn message exchange pattern. ChainBuilder ESB provides the reliable delivery option for message queuing and guaranteed delivery.  The fire and forget is similar to store and forward integration pattern.

- **Publish and subscribe**: ChainBuilder ESB provides support for the pub and sub messaging pattern via the JMS component.

# ChainBuilder ESB Services and Features

ChainBuilder ESB provides varieties of services via pre-built JBI binding components and service engines as well as the CCSL layer.

## Transport Protocols

Transport Protocols are supported via pre-built JBI binding components in ChainBuilder ESB. ChainBuilder ESB supports the following protocols:

- **File –** Uses data files as the source or target of messages.
- **FTP –** Uses FTP to process files over remote server. It supports basic mode like the File protocol and advanced XML-based scripting mode for flexibility and performance.
- **JMS –** Places or retrieves messages over JMS queues. Supports both JMS queues and JMS topics. The JMS component provides a reliable delivery mode for guaranteed message delivery.
- **TCP/IP –** Communicates between different types of computers and computer networks using TCP/IP (Transmission Control Protocol/Internet Protocol). It supports both client and server with SSL. It provides built-in handlers for CR LF, length-encoded and HL7 MLLP (minimum low layer protocol). Users can define their own protocol handlers as well.
- **HTTP(S) –** Supports secure HTTP client and server. Uses HTTP client to transmit messages to remote web servers. The HTTP server is built on top of Jetty to allow messages transferred from web-enabled application client.
- **SOAP over HTTP(S) –** Similar to the secure HTTP protocol. Enables interaction with SOAP-based web services and expose as a web services.
- **SMTP/POP3/IMAP –** Enables sending and receiving emails.
- **Script -** Develops custom communication protocols.

## Web Services

The support of web services is a fundamental concept in an enterprise server bus. ChainBuilder ESB provides comprehensive support for web services. There are three common use case scenarios in ChainBuilder ESB web services. First, it can be used to invoke the back-end web service as part of a process flow. Second, it can be configured as a proxy for back-end web services with additional routing, transformation and business rules. Third, it can be configured to expose the back-end legacy services, application and databases into a callable web service. ChainBuilder ESB supports web service standards like SOAP, SOAP with Attachment and MTOM, and the web service component is built on top of the Axis2 engine.

## Routing

The Content-Based Router (CBR) examines the message content and routes the message onto a different channel based on data contained in the message. The routing can be based on a number of criteria such as existence of fields, specific field values and message metadata. The CBR also supports a message split pattern by splitting an incoming message and routes to multiple destinations in parallel.

The Content-Based Router can identify the message using:

- HL7 – Based on the HL7 standard
- X12 – Based on X12 standard
- Fixed data
- Comma Separated Value (CSV)
- XPath
- Script - Use a custom program written in a JSR-223compliant scripting language, like Groovy.

The routing expression can be specified with:

- Regular expression
- XPath expression
- Exact match

## Transformation

Transformation is often needed when the message formats between source and destination are different. A map file is defined to translate data from one format to another. ChainBuilder ESB supports translation using the standard XSLT language as well as XML-based transformation language which includes features such as looping, iteration, condition, JDBC database operation and user operation to provide maximum power and flexibility.

A data mapping can be created between XML to XML, XML to non-XML, non-XML to XML and non-XML to non-XML mapping. A Map Editor tool is provided to allow users to easily create a map via a drag-and-drop visual interface.

## Data Normalization

Data normalization converts all inbound non-XML messages such as HL7, X12 EDI, and proprietary format from a transport into a XML format via a Parser component. Unlike other ESB products where normalization is required in the architecture, in ChainBuilder ESB, normalization is an option. In most integration scenarios implemented using ChainBuilder ESB, data normalization is not required since non-XML data can be transformed to non-XML data directly. But in some cases when the process starting from source endpoint focuses on the XML processing, data normalization should be used as a best practice to follow for performance and simplicity.

## Validation

ChainBuilder ESB provides the capability for validating incoming and outgoing messages for both XML data and non-XML data. XML data is validated using XML schema. Non-XML data is validated against the message format built from the ChainBuilder ESB Format Editor.

Message validation can be configured to occur at the inbound transport protocol or at the transformation in a process flow.  Validation can be applied independently at the source or the target at transformation.

## Security

ChainBuilder ESB provides a range of support for security including encryption, authentication, authorization, certificate management and role-based authorization. The following different aspects of security are supported in ChainBuilder ESB.

- **Transport-Level Security**: ChainBuilder uses SSL (Secure Socket Layer) to support confidentiality, message encryption and authentication for transport protocols of HTTP, TCP/IP, SMTP and POP3. It can be configured to use anonymous, basic authentication, client authentication and server authentication.
- **Message-Level Security**: ChainBuilder ESB PGP Service Engine supports message level security. The HTTP component supports the latest Web Service standards like WS-Security, SAML, and WS-Trust by passing the digitally-signed SOAP message. ChainBuilder ESB provides the infrastructure support to allow users or development partners to develop the Security Token Service (STS) to verify, sign and map security assertion.
- **User Management and Role-based Authorization**: The Admin Console supports adding users in different groups with different roles. The functionality provided by the Admin Console is based on user role.

## Transactional Support

ChainBuilder ESB supports transactions in the JDBC, ETL and the JMS components. In the JDBC component, ChainBuilder ESB accepts an XML-based message interface with actions of startTransaction, endTransaction, commit and rollback. In the JMS component, the commit or rollback of a transaction (when receiving message from a queue) is based on the result of the message exchange from the destination endpoint. In ETL component, the coordination of the transaction commit and rollback is based on the result of a message exchange and its metadata.

## Scheduling

A Unix Cron-like schedule can be defined for any polling-based source endpoint. This includes the File, FTP, ETL, Script, and custom components.  The built-in scheduling can retry until a certain condition is satisfied, and it can create a success or a failure message that routes to different endpoints for email notification.

## Scripting

ChainBuilder ESB supports scripting via JSR-223, using the ScriptEngine built within Java 6.  Java or any JSR-223 compatible scripting language can be used for scripting in ChainBuilder ESB. Groovy is the supported JRS-223 scripting language out-of-the-box.

## Reliable Delivery

Reliable delivery is an important requirement of an ESB when using the Fire and Forget messaging pattern. When a one-way message is received from a source endpoint, the message is always to be delivered or logged in case of errors.

ChainBuilder ESB uses the JMS component to accomplish reliable delivery. When a message is received from source endpoint, it is routed to a JMS component for reliable delivery. Built-in transactional retry logic in the JMS component is used based on the result from the target endpoint.  The message will be removed from the JMS queue only if it is delivered to the target.

## Industry Standards and Variants

ChainBuilder ESB provides data definition and translation support for industry standards like HL7 and X12. An entire set of data dictionaries is provided online, with all elements, fields, segments, and message layouts.

A **variant** is a copy of a message standard that uses the base definition of that standard to rearrange or augment it. The original message standard remains unchanged in the repository, for reference or to create new variants. Users can create their own extensions, if necessary. Enterprise-specific extensions can automate data translation runtime systems, or they can adapt incompatible "compliant systems" to one other.

## Alerts

ChainBuilder ESB provides the ability to define an alert based on a set of conditions. Users can define the severity levels for alerts including info, warning, error, critical and fatal. Multiple alerts can be combined to monitor the Service Level Agreement (SLA) for business. ChainBuilder ESB supports the following alert types:

- Server status
- Transaction count
- Error count
- Transaction per second
- Latency
- Service Assembly status
- Endpoint status
- Component status
- Component installation status
- JMS Dead Letter Queue
- JMS transaction retry count exceeded
- Endpoint last received
- Endpoint last sent
- File change status
- License expiration
- CPU usage
- Virtual memory usage

## Database Integration Services

ChainBuilder ESB provides extensive data integration services using the standard JDBC API. Users can define JDBC operations in a message transformation for online database lookup and update.

The JDBC component provides an XML-based messaging interface with embedded SQL to perform transactional database query and update. Users can control the beginning and end of a transaction as well as the commit and rollback via a messaging interface.

At the consumer endpoint, the ETL component provides the ability to generate a complex XML message from database queries based on a data source map. At the provider endpoint, it can be used to perform complex database updates from an XML message based on a data source map. There is no SQL coding required. A drag-and-drop visual tool is provided to create a data source map.

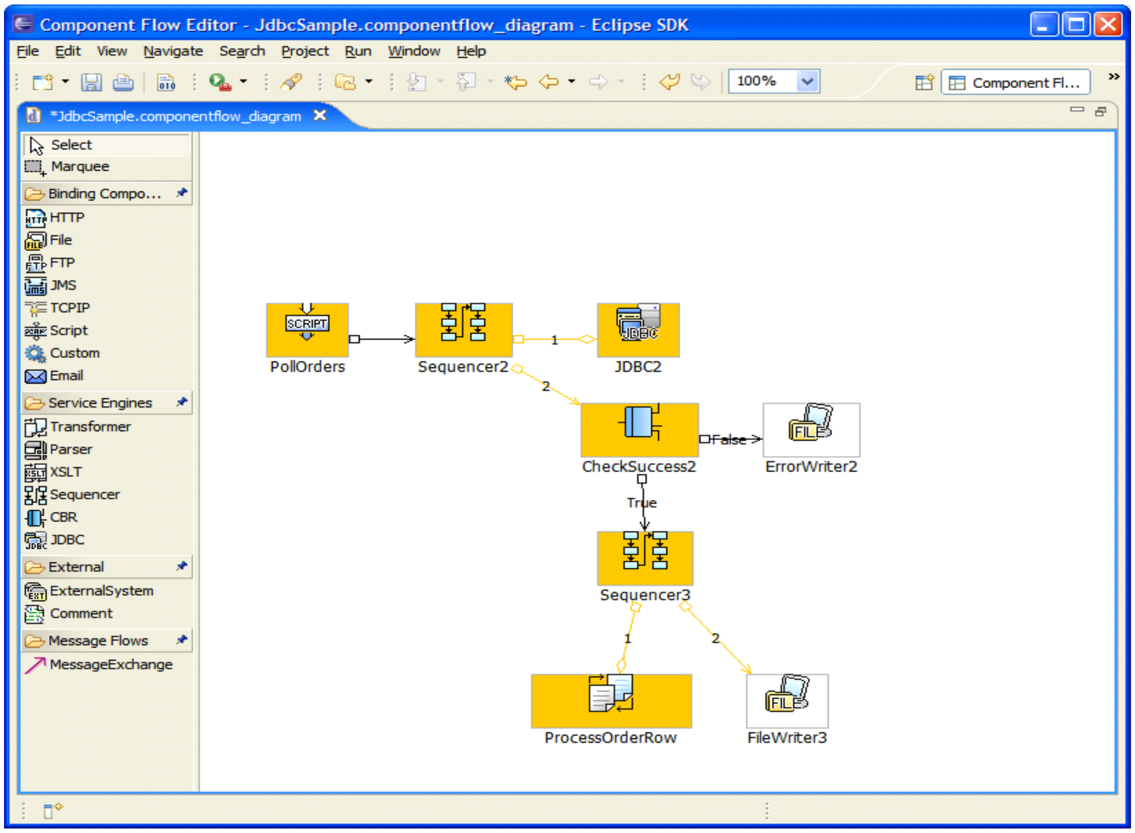
## Migration between Environments

ChainBuilder ESB provides macro support to ease the migration from one environment to another. The user-defined macros can be used virtually anywhere in the configuration including endpoint setting, map, file, lookup file. The macro is defined externally to the Service Assembly archive in a properties file. This allows settings to be changed without modifying and re-deploying the Service Assembly. This also makes it much easier to place settings for different environments (test, production, etc) in separate locations while using the same service assembly archive.

## ChainBuilder ESB Design Tools

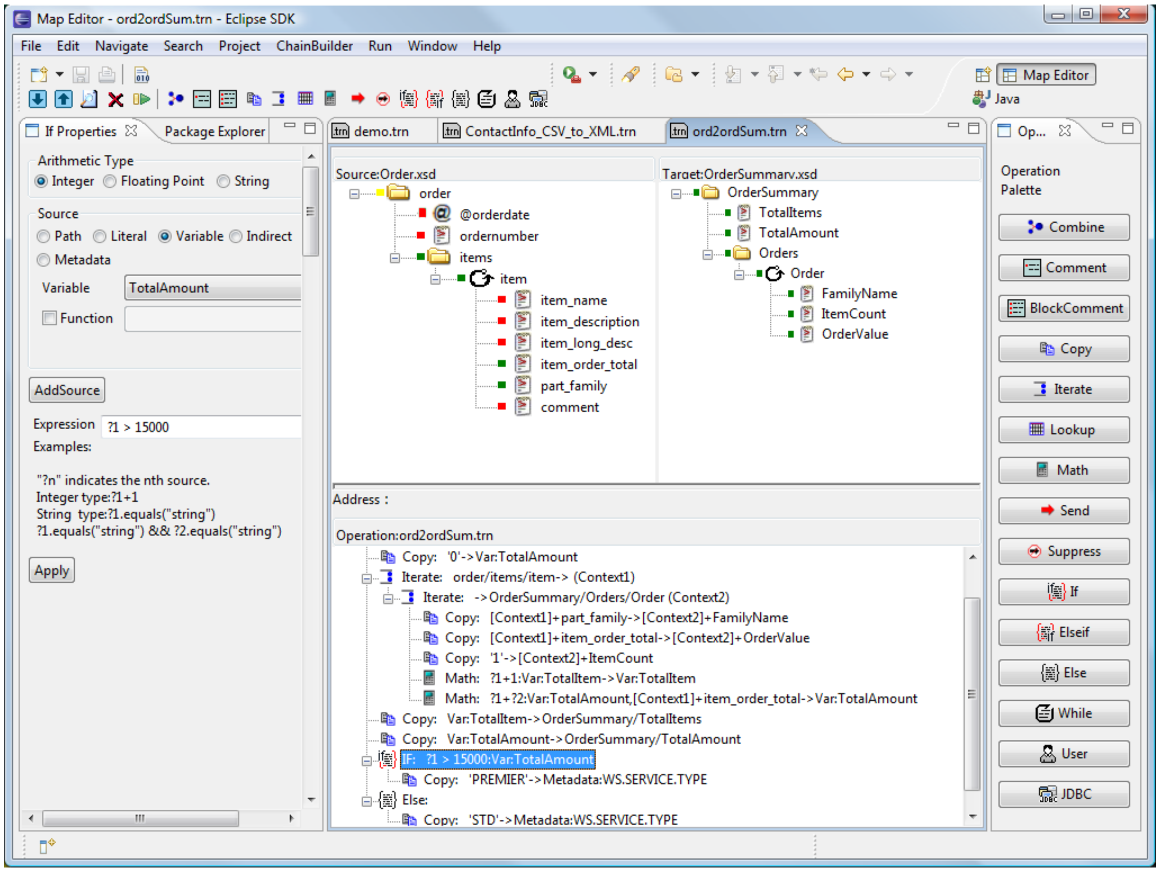
### Component Flow Editor

The Component Flow Editor is the process design tool for ChainBuilder ESB. It provides an easy to use graphical interface for creating JBI Service Assemblies. Using the popular Eclipse IDE platform, developers can create new and modify existing components through drag and drop, wizard, and fill-in forms functionality. The Custom Component Framework allows any user-defined JBI component to be configured within the flow.



## Map Editor

ChainBuilder ESB provides a powerful mapping tool to create a map from source to target for both XML and non-XML formats. The easy-to-use Map Editor allows users to use drag-and-drop to quickly create a complex map.



Arithmetic Type: [✓ Integer] Floating Point String

Source: Path Literal [✓ Variable] Indirect Metadata

Variable: TotalAmount

Function: (unchecked)

AddSource

Expression: ?1 > 15000

Examples:

"?n" indicates the nth source.
Integer type:?1+1
String type:?1.equals("string")
?1.equals("string") && ?2.equals("string")

Apply

```
Source:Order.xsd
order
    @orderdate
    ordernumber
    items
        item
            item_name
            item_description
            item_long_desc
            item_order_total
            part_family
            comment
```

```
Target:OrderSummary.xsd
OrderSummary
    TotalItems
    TotalAmount
    Orders
        Order
            FamilyName
            ItemCount
            OrderValue
```

Address :

```
Operation:ord2ordSum.trn
Copy:   '0'->Var:TotalAmount
Iterate:   order/items/item-> (Context1)
    Iterate:   ->OrderSummary/Orders/Order (Context2)
        Copy:   [Context1]+part_family->[Context2]+FamilyName
        Copy:   [Context1]+item_order_total->[Context2]+OrderValue
        Copy:   '1'->[Context2]+ItemCount
        Math:   ?1+1:Var:TotalItem->Var:TotalItem
        Math:   ?1+?2:Var:TotalAmount,[Context1]+item_order_total->Var:TotalAmount
Copy:   Var:TotalItem->OrderSummary/TotalItems
Copy:   Var:TotalAmount->OrderSummary/TotalAmount
IF:   ?1 > 15000:Var:TotalAmount
    Copy:   'PREMIER'->Metadata:WS.SERVICE.TYPE
Else:
    Copy:   'STD'->Metadata:WS.SERVICE.TYPE
```

The available map operations include:

| | | | |
|---|---|---|---|
| Copy | Combine | If/ElseIf/Else | While |
| Comment | Comment Block | Math | Iterate |
| Send | Suppress | Table Lookup | JDBC |
| User Operation | | | |

Users can define variables in operations and use a variable to dynamically evaluate a path in the source or target. The filter framework allows users to change the values before it is assigned to a target. The Map Editor also has access to all built-in metadata as well as user-defined metadata.

The Map Editor provides User Interface convenience features such copy, paste, cut, delete, search, short cut key, move, and synchronization to maximize developer productivity. It has an auto mapping feature to allow users to create a map with a single drag and drop, particularly beneficial if the source and target share a common structure.

## Format Editor

The Format Editor is the message format builder for ChainBuilder ESB. It allows users to create format definitions for fixed-record layout, variable-record layout (e.g, Comma Separated Values) as well more complex structured layout (e.g, HL7 or X12).

Depending on the type of message layout, users can specify the following in the Format Editor:

- Data type
- Length
- Offset
- Prefix character
- Fill character
- Delimiter
- Escape character
- Tag

The format file saved by the Format Editor can be used by the Map Editor to create a map.

## Variant Editors

The Variant Editor allows users to view or create a variation based on the data directory for industry standards. ChainBuilder ESB has two Variant Editors: the HL7 Editor and the X12 Editor.

In the HL7 Editor, users can define fields, data segments, and messages for the HL7 standard. In the X12 Editor, users can define data elements, composite data structures, data segments, and transaction sets used by the X12 standards.

## Testing Tools

Testing tools in ChainBuilder ESB allow users to run a test on a map, on an HL7/X12 variant, or on a message format file without starting the ChainBuilder ESB runtime environment. A graphical user interface displays the test result.

## WSDL Designer

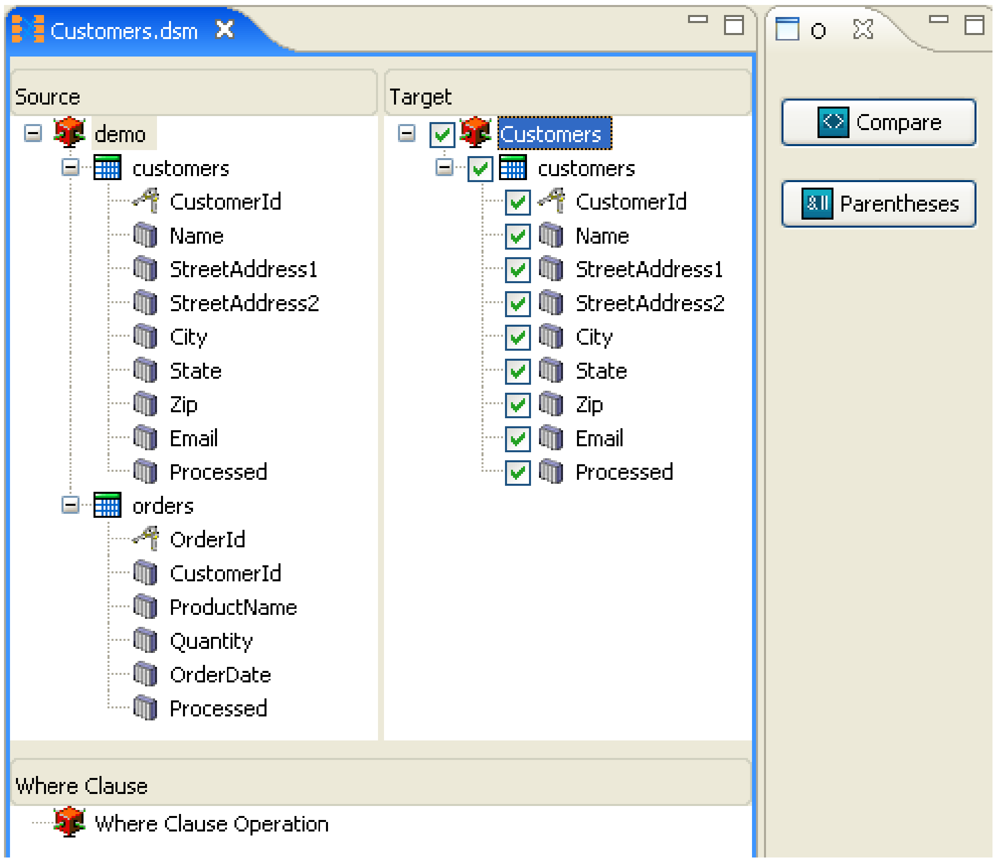
The WSDL Designer allows users to build a WSDL definition by importing an existing schema. It hides the complexity of hand-coding a WSDL file. A common usage scenario allows user to specify the WSDL file for the HTTP component to expose a web service.

## WSDL Importer

The WSDL Importer is a tool to import an existing WSDL and create the XML schema from it. When consuming a web service, users are responsible for creating a correct SOAP message. With the generated XML schema, users can use the Map Editor to easily create a map for it.

## ETL Design Studio

ChainBuilder ESB's ETL Design Studio is a set of Eclipse plug-ins that utilize wizard-driven configuration screens to walk the user through the process of connecting to a database, and the drag-and-drop selection of tables and fields to be extracted or loaded.



The ETL Design Studio wizard finishes by generating an XML schema, defining either the format of data to be extracted from the database or the format used for inserts or updates.  Advanced settings, such as transaction and batch size, can be configured to achieve maximum throughput performance.

After data extraction, the ChainBuilder ESB Map Editor can be used to perform data mapping, validation and data cleansing before data is loaded into target data source.

## Certificate Management

The Certificate Manager Wizard provides the ability to maintain keys and certificates used to establish an SSL connection. The wizard provides the ability to import and export certificates to/from a Key Store or Trust Store. The Certificate Manager Wizard creates self-signed certificates, issues Certificate Signing Requests, and provides viewing of the contents of certificate files and Key Store files.

The wizard guides users through the common scenarios of anonymous, self-signed, and Certificate Authority (CA) signed certificate.

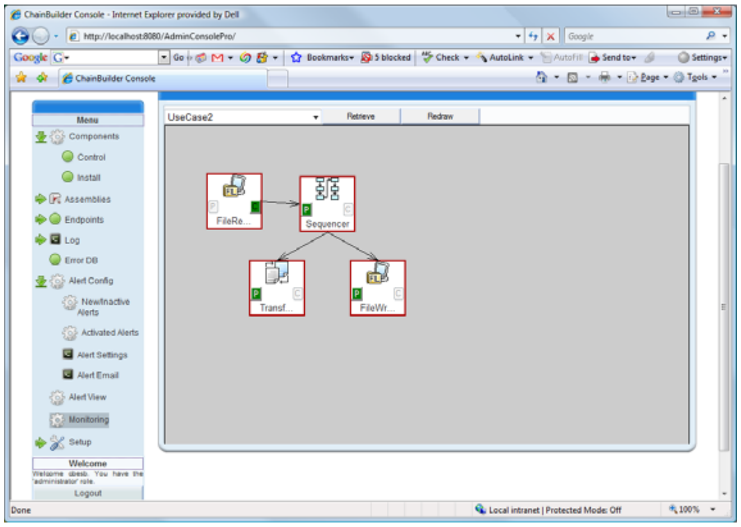
## ChainBuilder ESB Control and Management

The ChainBuilder ESB Admin Console is a web-based application that provides control and management of the ChainBuilder ESB server.

### Monitoring and Control

The Admin Console provides the following control and management capability:

- Install and remove JBI components and shared libraries
- Install and remove ChainBuilder ESB JBI Service Assemblies
- Start and stop components and shared libraries
- Start and stop ChainBuilder ESB JBI Service Assemblies or Endpoints
- View and change ESB endpoint settings
- Graphic view of ChainBuilder ESB JBI Service Assembly



## Statistics View

The Statistics view shows the endpoint-level statistics information like message count, transaction per second and message latency.

## Error Diagnosis

The Error Database view gives users the ability to view the detailed error information when an error occurs in ChainBuilder ESB. It helps users trouble shoot runtime issues. Original messages are saved in the error database to allow users to reprocess them.

## Alert Configuration and View

Alert functionality in ChainBuilder ESB allows the user to define varieties of conditions and the desired notification mechanism if the conditions are met.

The ChainBuilder ESB Admin Console allows users to create one or more alert definition files as well as to define notification actions (e,g, to execute a command, Email and JMS). It allows users to activate or de-activate alert definitions.

When an Alert definition is activated, the alert engine monitors the conditions being checked and triggers an alert if the conditions are met. Users can use the Admin Console to view, archive or delete triggered alerts.

## User Management

The ChainBuilder ESB Admin Console provides user management to allow an administrator to add additional users with different roles.

The available roles are:

- **Administrator**: A “Super User”. Admin users can add/delete/modify users. Administrators can perform all functionalities allowed to the Operator or to the “normal” user
- **Operator**: Administer the entire ChainBuilder ESB, but without the ability to create new users.
- **User**: Can only view information. Users can not perform any action such as delete, update, start, stop, or uninstall.

## Command-Line Tools

ChainBuilder ESB provides a set of command line utilities to allow developers to perform deployment and testing without running the entire server environment. It also allows system administrators to perform administration such as viewing the audit log, viewing runtime database and user database or perform maintenance to JMS queue for reliable delivery.

## Supported Environment

The following is the supported and recommended environment for ChainBuilder ESB:

- **Operating Systems**: Windows and Linux (x86/GTK2)
- **System Requirements**: Pentium 4 with minimun1GB RAM and 10GB hard disk space; Dual 3.0Ghz with 2GB RAM preferred
- **Java** : JDK 1.5 and JDK 1.6
- **Languages**: English and Chinese

Since ChainBuilder ESB IDE is built on Eclipse and the runtimes is built using 100% Java, ChainBuilder ESB can run on any platform that has JDK 5 or 6.

## References

1. Bostech Company homepage: http://www.bostechcorp.com
2. ChainBuilder ESB community site: http://www.chainforge.net
3. JBI Specification: http://www.jcp.org/en/jsr/detail?id=208
4. JSR-223: http://www.jcp.org/en/jsr/detail?id=223
5. Groovy: http://groovy.codehaus.org/

## For More Information

To learn more about the ChainBuilder ESB, please send your inquiry to Bostech Corporation at info@bostechcorp.com, or visit the Bostech Contact Us page at: http://www.bostechcorp.com/ContactUs.