EDImation Guide

Infinite Functions, Inc.
PO Box 240
Creston, CA 93432
USA

+1 (805) 591 4578

Email: sales@infinitefunctions.com

Please see us on the Web at:
www.infinitefunctions.com
www.floormation.com
www.edirmation.com

Edited: October 29, 2019

Table of Contents

GENERAL INFORMATION ON FLOORMATION EDI INTERFACE

The EDI module (EDImation) provides a fully automated platform for coordinating bi-directional communication between Floormation and a site's local systems or the company's enterprise systems or external partners. It provides scheduling and mapping services as a tightly integrated, scalable solution and interoperates with industry standard protocols.

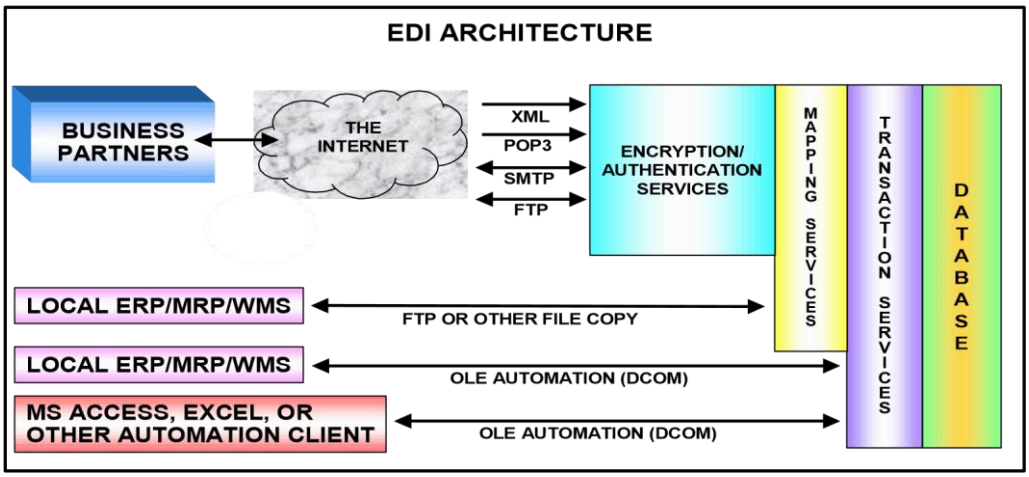
EDImation is comprised of the following services:

Mapping Services:	A flexible, JavaScript-enabled mapping environment is used to translate data between standard formats (ANSI X12, UN/EDIFACT, XML, SAP IDOC, etc.) and the internal representation required by Floormation.
Scheduling Services:	An integrated Floormation system server allows scheduling EDI processing to occur at predetermined intervals throughout the day. The scheduler can be used to perform both EDI and non-EDI-based tasks.
GUI Interface:	All EDI processing logs, schedules, partnerships, and other EDI configuration information are accessible from Floormation's standard graphical interface.
JavaScript:	The entire EDI system is controlled by an embedded JavaScript execution environment. Within the JavaScript run-time, all information associated with the EDI system is available for inspection and manipulation. All services (transport, encryption, mapping, scheduling) are exposed to JavaScript through an EDI object model.

The EDI module was designed to provide a scalable, robust processing environment. A typical configuration might use one server to host the database, a second to host the application server and a third to host the EDI servers.

Additional EDI servers can be added as the EDI service usage increases. The additional machines can be dedicated to servicing particular trading partners or a load-balancing scheme can be used to distribute processing load.

EDI Architecture:



Choosing a Transport Service:

When choosing which transport services to use, many factors must be considered. Existing or perspective trading partners may have established EDI programs and will prefer one method to another. Reliability and ease of installation and maintenance should be strongly considered. Transport latency requirements should be carefully reviewed to ensure the EDI program would meet performance requirements. Finally, the type

of physical communication link (always on Internet, on-demand/dial-up Internet, private network or leased line, dial-up modem, etc.) directly limits your choices.

Different transport services can be used with each trading partner, although companies who are too accommodating in this regard will incur increased support costs. It is always easier to support one transport mechanism than many.

It is also possible to use different transports for a single trading partner, including receiving using one transport and sending using another or using one transport as the primary and another as the backup when the primary fails. Partnership information is stored in a single location and is relatively transport independent. ("Relatively" due to the encryption services associated with each partnership. Private-key encryption cannot be used with SMTP/POP3 transports, as those are limited to S/MIME encapsulation.)

The following is a summary of each transport and its features.

FTP:

Physical Link

Real-time receiving requires an always-on network/Internet connection.
Batch-mode receiving can use an always-on or dial-up network/Internet connection. Real-time sending can use an always-on or dial-up network/Internet connection although the dial-up latency must be considered.
Batch-mode sending can use an always-on or dial-up network/Internet connection.

Guaranteed Messaging

Fully integrated.

Encryption services

None, private-key, or S/MIME.

SMTP:

Physical Link

Real-time receiving requires an always-on network/Internet connection.
Batch-mode receiving is not possible.
Real-time sending can use an always-on or dial-up network/Internet connection although the dial-up latency must be considered.
Batch-mode sending can use an always-on or dial-up network/Internet connection.

Guaranteed Messaging

Fully integrated.

Encryption services

None or S/MIME. (Private-key encryption cannot be used due to its binary format.)

POP3:

Physical Link

Real-time receiving is not possible.
Batch-mode receiving can use an always-on or dial-up network/Internet connection.
Real-time sending is not possible.
Batch-mode sending is not possible.

Guaranteed Messaging

Fully integrated.

Encryption services

None or S/MIME. (Private-key encryption cannot be used due to its binary format.)

Other File Transfer:

Physical Link

Real-time receiving is not possible.
Batch-mode receiving is possible.
Real-time sending is not possible.
Batch-mode sending is possible.

Guaranteed Messaging

Not integrated but can be customized either by IF or site's IT department.

Encryption services

None, private-key or S/MIME. Encryption choice may be limited by the transport (e.g. VANs may not allow it).

Mapping Services:

Transaction mapping is the process of converting a document from one format to another. For inbound EDI transactions, a file containing the EDI data (typically in a format dictated by ANSI X12 or UN/EDIFACT) is converted to a format usable by the JavaScript run-time. For outgoing EDI transactions, the data is converted from the JavaScript representation to an external file format.

A transaction map or map for short, is simply a file containing the description of an EDI transaction format. The map contains formatting descriptions such as record and field sizes, loops and delimiters. It also may contain descriptions used to validate data. These descriptions can consist of either literal values (i.e. the string "XYZ" must occur at this location) or regular expressions. Regular expressions (REs) provide a means of describing data whose general format is known but actual value is not. For example, an RE can be used to indicate that the current field must contain only numeric values, or a date, or timestamp, or just about anything else that has recognizable format.

The process of using a map to read or write a transaction file is called mapping. During mapping, three different items are being referenced and/or generated: the map file, the transaction file and the data model. The data model is the internal JavaScript representation of the transaction file and allows a JavaScript program to read and write fields in the file using native object and property references rather than field offsets and lengths.

The data model is what makes mapping so powerful. Because the physical layout of the file is hidden from the JavaScript program, the developer of a transaction has to only concentrate on the logical format and content. Additionally, because the physical layout is encapsulated, a single JavaScript interface can be designed to work with many different versions of the same transaction. So Vendor-A can use version 3040 of the X12 943; and Vendor-B can use version 3070; and Vendor-C can leverage its already developed version 3050.

The process of mapping involves two different scenarios. When using a map to read a file, the file's contents are compared against the map's format and content descriptions. If everything matches, the file is deemed valid and the resulting data model is ready for inbound EDI processing. When using a map to write a file, the data model generated by the EDI outbound processing is compared against the map's content description and, if it matches, is formatted and output according to the map's format description.

File Formats:

In general, programmatically readable files fall into one of three different types: fixed-length, delimited, and tokenized.

- A fixed-length file uses predefined field and record sizes. When a value is less than its corresponding field length, it is padded, either to the left or right, with the appropriate pad character. Likewise, when a record's contents are less than the defined length, it is right-padded. In all cases, the pad characters may need to be defined globally, record-by-record and/or field-by-field.
- A delimited file uses one or more characters with special meaning to denote the ends of fields, records, and other elements. When one of the special characters must appear as part of a value, an "escape" character must be used to turn-off the delimiter character's special meaning. The delimiter and escape

characters may be predefined, defined by usage or context, or defined record-by-record and/or field-byfield.

- A tokenized file like XML consists of predefined symbols and symbol classes, typically separated by zero or more occurrences of special characters called "whitespace." An example of a predefined symbol might include punctuation characters like '.' (dot) and '&' (ampersand) and multi-character punctuation like '...' and '&&'. Symbol classes are tokens that have describable content but whose values are not predefined; for example, an identifier token might consist of a letter followed by any number of letters and digits. Whitespace is "syntactic filling." It is not part of the actual content of the file but is used to aid readability (by humans) and is only required when two adjoining tokens would be confused as one if there was nothing separating them. The term whitespace refers to the use of space characters (the "white" between words) to separate tokens in a language such as C++ or English but in practice, can be defined as any character or set of characters.
- EDImation includes its own XML sub-module

Of course, there is no hard-and-fast rule that a file must consist of only one type of formatting. A file layout could utilize two or all three types of formatting and switch among them based on position or other contextual information.

The Floormation mapping services can handle all three types of file formats. Fixed-length fields and records are specified using simple length and padding attributes. Delimited files are handled by specifying delimiter and escape characters at the file and record levels (delimiters are meaningless at the field level). Tokenized files are parsed using regular expressions to describe the predefined symbols and symbol classes and a special attribute at the file level defines the whitespace character set.

Transaction Scripts:

Once a map has been created that describes the content of a transaction file, a JavaScript program is required to do something intelligent with it. For inbound transactions, the JavaScript is responsible for reading the data model and either formatting and submitting the data to the Floormation application server (recommended) or modifying the database directly (not recommended). For outbound transactions, the JavaScript is responsible for populating the data model.

The inbound scripts tend to be fairly generic and often can be written to handle multiple trading partners. Outbound scripts are more likely to be partner-specific unless the transaction is fairly common and used by multiple partners.

Scheduling Services:

Integral to the EDI architecture is a scheduling service that allows tasks – both EDI and non-EDI based – to be executed at set times during the day, week, month and/or year. The scheduled tasks can be used to initiate inbound and outbound EDI transaction processing, perform database cleanup and maintenance, send HTML reports to distribution lines, or just about anything else.

Each task is written in JavaScript and executes using the standard EDI Object Model.

Floormation's standard graphical interface is used for adding, updating and deleting tasks from the Scheduler. The screen Scheduler under the EDI section of the main menu brings up the configuration screen.

EDI CONFIGURATION

Floormation Configuration for EDI Interfaces

The Floormation EDI Server is a powerful and flexible tool for moving data into and out of Floormation. Configuring Floormation to process EDI Jobs is a simple task outlined here.

Launching the Floormation EDI Server

The Floormation EDI Server is controlled by the Floormation Apache module, mod_fmapp.so. To include the EDI Server in the suite of servers controlled by the Server Manager, edit the fm6.conf file in the /etc/httpd/conf directory so that it contains the line:

fmedi –dom /<domain>

Where <domain> is the name of the Floormation Domain that the EDI Server will serve. A Floormation Domain is used by Floormation Servers to redirect URL requests to file paths. For example, in the URL 'http://10.10.10.10:2020/DOM1/winui.jsh', the Floormation Domain 'DOM1' allows Floormation to specify the file 'c:\fm6\app\winui.jsh'. Floormation Domains also determine Parameter Substitution: if the fm6.conf file contains the parameters 'DOM1/FMDSN ORC1' and 'DOM2/FMDSN ORC2', a request to Domain DOM1 would resolve the parameter 'FMDSN' to the value 'ORC1'.

To verify that EDI is running use ps –ef | grep –I fmedi

Control by Floormation App

The fmedi process is a cron-like scheduler purpose built to work with Flooramtion databasestored schedules and the Apache framework. The process is launched automatically when the Apache module, mod_fmapp.so, starts and terminates automatically when Apache stops. A monitoring thread inside the primordal Apache process maintains a socket connection with the fmedi process. If the Apache thread sees a disconnect (from fmedi aborting), it automatically launches a new fmedi process. If fmedi sees a disconnect (httpd stop), it automatically kills its self.

Configuring OS Directories for EDI

Floormation EDI Jobs read from and write to Operating System directories. The Floormation EDI Server uses the following directories, all of which can be created under an EDI directory: **archive:** This directory stores unmodified data files on the Floormation Server. **inbound:** This directory is the arrival point for data files. **jse:** This directory holds EDI Java Script programs (*.jse) **log:** This directory holds a process record of each EDI Job, which contains at least the start time and completion time of the job. **map:** This directory contains the data map files for inbound EDI Jobs (*.sch) **outbound:** This directory contains outbound data files. **outtemp:** This is a working directory for outbound EDI Jobs. **trc:** This directory contains a record of the parsing of inbound data files by the schema files.

Configuring Floormation Map Files for EDI

The Floormation configuration files /etc/httpd/conf/fm6.conf must be configured so that the Floormation Server has the permission to write and knows where to find EDI information. Add the following

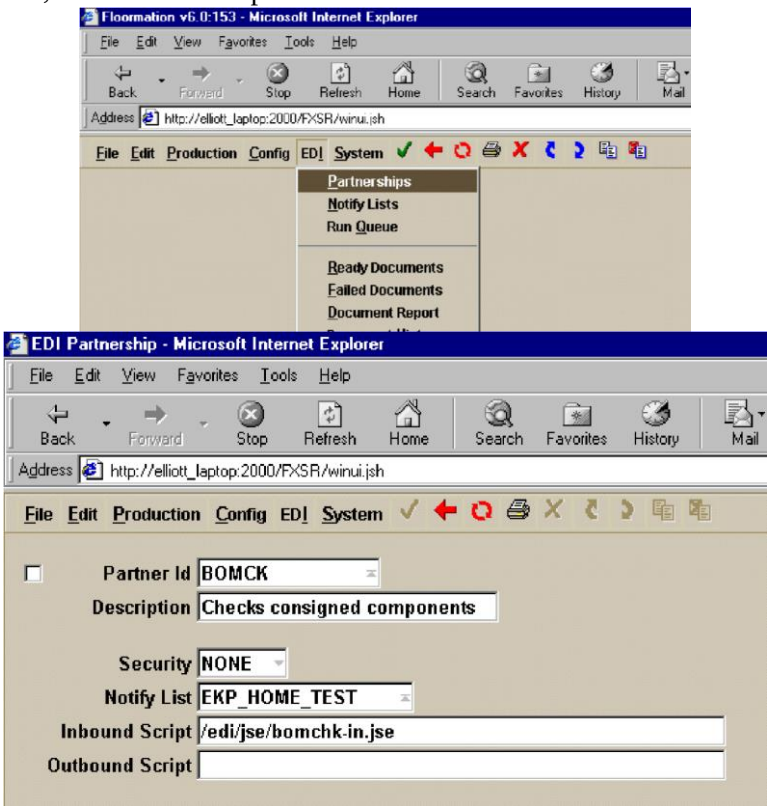
```
/<domain>/edi     <floormation os directory>/edi      write
/<domain>/log     <floormation os directory>/edi/log  write
/<domain>/trc     <floormation os directory>/trc      write


/FXSR/FMMAILHOST        <mail relay>
```

The parameter FMMAILHOST is called by Java Script EDI programs. It refers to the mail server through which the Floormation EDI Server will send email notification of the status of EDI Jobs. Since it is called in Java Script and not by the server, the name of the parameter can be changed.

Configuring an EDI Partnership via Internet Explorer

From the EDI Menu, select Partnerships.



The fields in the picture above are the minimum required to configure an EDI Partnership in Floormation. The data fields on the EDI -> Partnerships screen that are not shown in the picture above are related to data encryption. If the Security field is set to 'NONE' they are not required.

Partner Id: This is an identifier for the EDI Partnership. When a data file is sent to Floormation, it needs to be identified with the value in this field so that the correct EDI Job can process the data.

Description: This is simply a description of the Partnership.

Security: This setting defines the encryption method that Floormation will use for transmission and reception of data. As mentioned above, the simplest setting is 'NONE'.

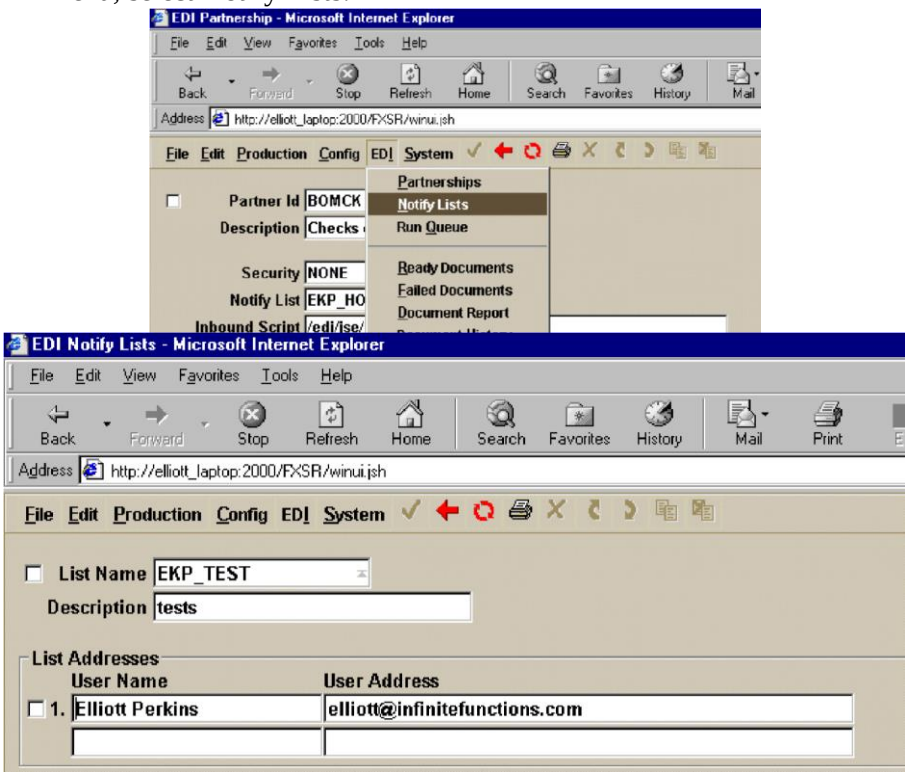
Notify List: This is the identifier for a list of email addresses to which Floormation will send messages if it is instructed to by the Java Script EDI Program associated with this Partnership.

Inbound Script: This is the path and name of the Java Script Program that will handle inbound transactions for this partnership. Note that the path is relative to the Floormation Top-Level Domain, and not to the Operating System File Structure. This script will be called within one minute when a file identified with this Partnership is sent to Floormation.

Outbound Script: This is the path and name of the Java Script Program that will handle outbound transactions for this partnership. Note that the path is relative to the Floormation TopLevel Domain, and not to the Operating System File Structure.

Configuring an EDI Notification List via Internet Explorer

From the EDI Menu, select Notify Lists.

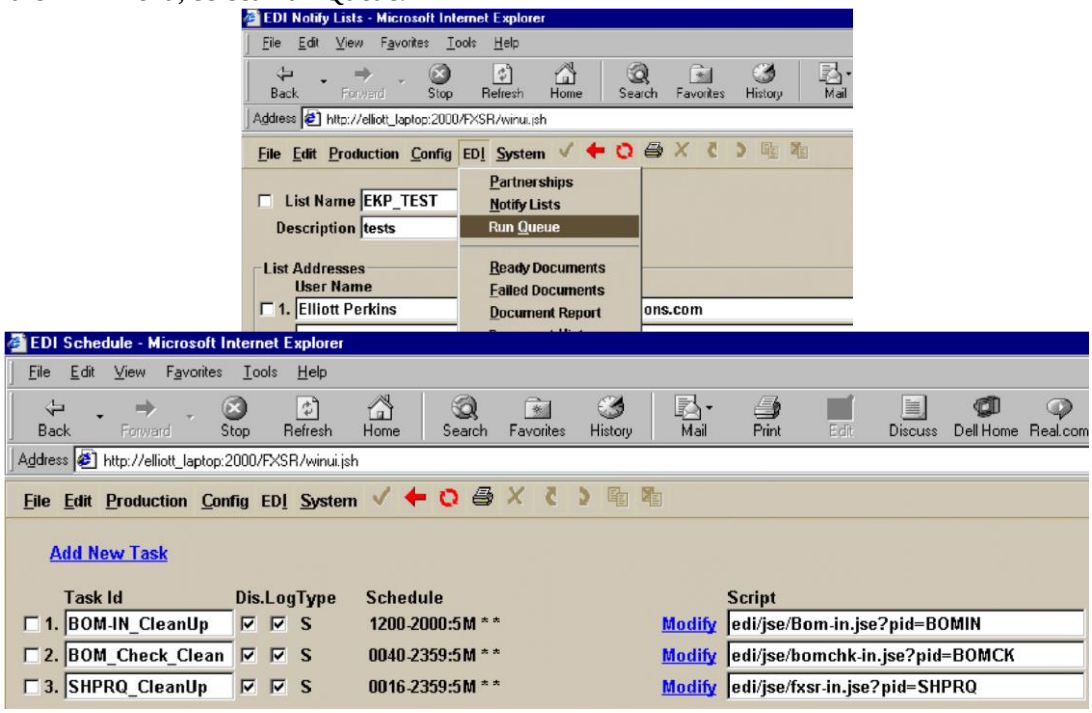


Any number of Notify Lists can be configured from this screen. Notify Lists are not bound to EDI Partnerships until a notify list is selected on the EDI Partnership Page. So, for example, two lists can be configured for one partnership: PID_DEV and PID_PROD. While the partnership interface is in development, the list can be set to PID_DEV, sparing the people on the Production List from receiving unwanted development-related email. When the partnership interface is in production, the PID_PROD list can be selected, ensuring that the right people know what is happening with the EDI Partnership.

Note: Since the Floormation EDI Server does not have its own mail server, any mail interface must be subject to the authentication rules of the relay server that is sending the mail. In some cases, this means that the Sender Address is validated. The sender address is set in the Java Script EDI Program.

Scheduling an EDI Task via Internet Explorer

On the EDI Menu, select Run Queue.



The EDI Schedule Page displays a list of scheduled tasks and allows users to add new tasks or modify existing tasks. Existing tasks are displayed in a table, with each row containing information about one EDI Task. The information is:

Task Id: This field is an identifier for the task.

Dis.: A check mark in this checkbox indicates that the task is disabled. If the task is disabled, it will not run.

Log: A check mark in this checkbox indicates that the task will record a log of its activities in the EDI Log directory, as defined in param.map and request.map.

Type: This field contains one of **S, A, O**, and indicates the type of the task. If the type is **S**, the task is run on a schedule, which is defined as blocks of time when the task can run, and a frequency within those times when it will run. If the type is **O**, the task is run once, at a particular date and time. If the task is type **A**, it is run after another task completes.

The type field can be modified using the **Modify** hyperlink on the same table row as the task.

Schedule: This field indicates the schedule on which the task will run. The schedule is easily configured using Floormation Pages. The format of the data on this row depends on the Task Type. If the type is **S**, the schedule has the format `HHMM-HHMM:FREQ *|D[,D...] *|M[,M...]`

Where HHMM-HHMM represents a starting time and an ending time, in 24-hour notation, and FREQ represents a frequency. * |D[,D...] represents either every day (*), or a list of days during which the task will run. Days of the week are prefixed with 'w'. *|M[,M...] represents either every month (**), or a list of months during which the task will run. For example 1200-1500:5M 1,3 9 will run the task from noon to three PM on the first and third of September.

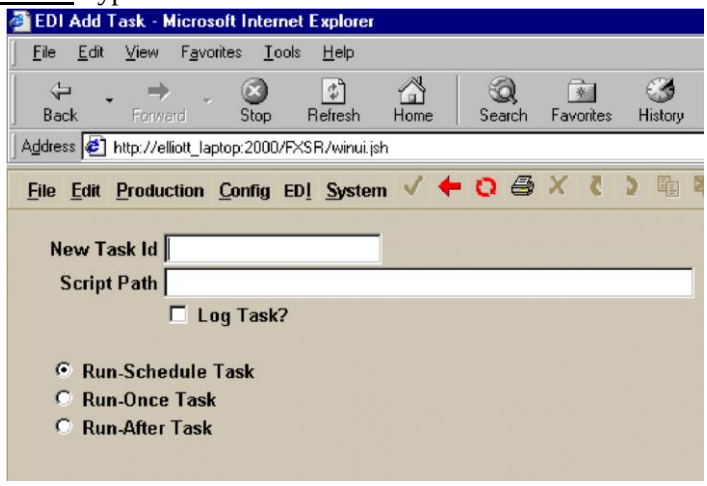
If the task type is **O**, the schedule has the format `YYYY-MM-DD HHMM`

Where YYYY is a four-digit year, MM is a two digit month, where January is '01', and DD is the day of the month, starting with '01'. HHMM is the time of day, in 24-hour notation. If the task type is **A**, the schedule is the name of the EDI Task after which this task will run. **Modify:** This is a hyperlink to Floormation pages where both the task type and the task schedule can be changed.

Script: This data field contains the location and name of the Java Script Program that will perform the EDI Task. Note that the path is relative to the top-level Floormation Domain, not the operating system's file structure. Arguments to the Java Script file are given with the syntax `script-path?<arg1>=<val1>;...<argn>=<valn>`

To add a new task:

Click the **Add New Task** hyperlink.



Enter the appropriate data in the data fields on the page.
New Task Id: This is the identifier for the new task.
Script Path: This is the path to the Java Script Program that will perform the EDI task.
Log Task?: A check in this checkbox indicates that this task will write to a log file.
Radio Buttons: Run-Schedule Task, Run-Once Task, Run-After Task: Select the Task Type for this task.
When the data is entered, save the Page. Floormation will refresh the screen with a page that allows the user to configure the schedule based on the Task Type.

To Modify an Existing Task:

Click the **Modify** hyperlink that corresponds to the task you want to modify. Floormation will present a page that allows you to select the task type. When you save this page, Floormation will refresh the screen with a page that allows you to configure the schedule based on the Task Type you have selected.

A Note on Data Processing for Scheduled Tasks:

In order to force the Floormation EDI Schedule to be rebuilt, an Oracle Sequence must be incremented. This is not necessary when the update is done from Internet Explorer, as the server scripting performs this task automatically. To update the EDI Schedule from an interactive SQL prompt, though, the following SQL must be processed in order to reload the schedule: `SELECT EDISCHSEQ.NEXTVAL FROM DUAL;`

If the above SQL is not processed, the schedule will not be updated.

Floormation Files used in EDI Interfaces

There are two types of files used for EDI interfaces: map files and Java Script files. The map files parse data files and load the data into Java Script arrays. These arrays are then used by Java Script files to update Floormation to reflect the new information.

Data Files

Data files are sent to Floormation in an agreed upon format, and contain the information, such as orders, their products and quantities, that Floormation needs to schedule its activities.

Map Files: Parsing Inbound Data

Map Files, with the extension .sch, are used by the Floormation EDI Server to parse inbound data files. They are written in a language that allows them to read data based on pattern matching or on field length.

Java Script Files: Performing EDI Interactions

Java Script Programs, with the extension .jse, are run by the Floormation EDI Server to process the data taken from data files by map files. Java Script files are responsible for all logical decisions that relate to data validation and insertion. As such, they must be tightly written to handle all manner of exceptions without corrupting the data in Floormation. Also, they are responsible for notification in the event of error, so the output must be sensible and robust.

The Java Script Environment and Floormation EDI

fmedi does not host javascript. It does only the following:

- monitors the task schedule
- maintains a queue of pending tasks (scheduled-tasks and run-now tasks)
- provides a pool of worker threads to submit the pending-tasks

When a task is ready to execute, it is added to a pending queue inside the fmedi process. A pool of worker threads (currently set at 8) grabs each pending task and does an HTTP GET to the Apache server with the task's URL. These tasks currently run as user 'SUPPORT' but we may need to add a user-id column to each task to allow tasks to run as different users/capabilities. An important feature if we want the task to be limited by our partner filtering, for example. Power users could be given the ability to configure a task to run as anyone, lower users could only schedule jobs (reports) to run as themselves.

The HTTP protocol allows the client (IE6 or fmedi, for example) to add HTTP headers to the request that provide contextual information. (These headers are visible to our javascript as the global object 'request'.) We could program fmedi to add specific headers that let our javascript know that fmedi is submitting the request and what to do with the output. Some possibilities:

- mail the output to a dist list
- save the output to a log file
- record the execution in a database log- ???

The user-agent string (request.UserAgent) contains the string "[fmedi]" at its very end. Specifically, the user-agent string submitted by fmedi looks like:

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0) [fmedi]

Whereas the user-agent string from IE6 running on win2k looks like:

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)

Once the HTTP GET is complete, the scheduler throws away any output. This is important - there is no hard-coded functionality in the scheduler anymore with the exception of the actual task scheduling logic. All logic now resides in the fmapp javascript and can be changed and modeled to suit a particular client's needs. We need to develop a general framework that will fit most clients.

The task thread that submitted the HTTP GET inspects the returned status code from the server to determine what to do next:

HTTP 200 is the only successful return status.
HTTP 100 'Continue' indicates a failure but nothing to worry about.
Any other status code indicates a failure and will be logged to a task exception table that records the task-id, URL as submitted, status code, and status message.

From the javascript side of things, you return statuses other than HTTP 200 to a client using the abort() and abortEx() functions. abort(string) is equivalent to abortEx(500, string).

If the task was successful (HTTP 200), any tasks defined to run-after it will be added to the pending task queue.

If the task was a scheduled-task (as opposed to a run-now), it's next run time will be calculated and added to the task schedule.

If the task was a run-now, it's record is removed from the database schedule table.

The task thread has now finished execution of that task and will return to the pending-task queue to wait for its next task.

To launch a Java Script program as a scheduled task, follow the instructions above for scheduling an EDI Task via Internet Explorer.

A record of the EDI Servers attempt to use a map file to parse the inbound file will be stored in the edi/trc directory, with a filename specified in the Java Script program, typically the PatnerId, the data file name, and the extension trc, bomck0001145.trc, for example.
A record of the Java Script program's execution will be written in the edi/log directory. If the program is launched as a scheduled task, the log file will be the task name and the three letter code for the day of the week, BOMIN_Clean_Up.thu, for example. If the scheduled task is run more than once on that day, the task will append to the log file. It will not overwrite the log file.

Writing to the log File from Java Script

Use the 'print(sString)' function to write to the log file.

DEVELOPMENT GUIDELINES FOR INBOUND EDI TRANSACTIONS

This document is intended to provide a step-by-step process for designing and programming Inbound EDI Transactions. This is a procedure manual, and it does not document any existing EDI Transactions.
EDI is an important if hidden part of any Floormation instance. Inbound EDI Transactions take data from a file provided by the customer and translate it into a format that is useful to the Floormation Application. This means taking a flat file, parsing useful data from it, validating that data against data in Floormation, and executing SQL statements that will put the data in the Floormation Database.
Inbound EDI Transactions validate the data before they put anything into the database. It is vitally important to understand that the scripts for inbound EDI Transactions are gateways into the applications, and that they cannot allow all data to pass.
It is vitally important that these scripts handle conditions where the data cannot be entered. This means sending email in the case of exceptions, and it means that the email has the reason that the data could not be entered clearly explained.
What follows is a procedure for designing and implementing inbound EDI Transactions.

Schedule

An EDI project involves effort and deliverable material from people in our organization and from people in the customer organization. It is important to be aware of the schedule at the beginning of the process, and as it evolves over the course of the project.

Generally, the first deliverable for an EDI Transaction is Integration Testing. The process should be programmed and tested before the Integration Test. This means identifying the date of the integration test, planning internal tests for the process, estimating the time those test will take, and working backwards from the date of the Integration Test to determine an internal deadline for the project.

For example, if the Integration Test is on the tenth of the month, and there are two days worth of testing, the internal development deadline is the seventh of the month, leaving the eighth and ninth for testing.

Design

Make Contact Lists

All EDI Transactions involve at least two parties, the sender and the receiver. Often, as the transaction developer, you are a third party. At the start of the job, identify the involved parties. Often, you will be in contact with a project manager working for a customer of Infinite Functions. This person will be the primary contact for the job. There will probably also be people on the customer's side who have tasks specifically related to the EDI Transaction; they will be in charge of configuring the interfaces and transmitting data. It is important to know who these people are. Begin the EDI job be making an annotated contact list, even if it is on paper.

Study the Process

All Inbound EDI Transactions are intended to automate some form of Data Entry. At the beginning of the design process, it is important to understand exactly what this means to the end user. For example, if the EDI transaction is designed to put work orders into a manufacturing system, consider what will happen the orders as the manufacturing supervisor will see them. In this example, the supervisor will want to schedule the order to production. Depending on the condition of product configuration and the bill of materials in Floormation, the supervisor may be able to double-click and schedule, or they may have to do some maintenance before they can schedule the order. In this example, it is a part of the EDI transaction to determine the Status Code of the work order when it is inserted so that the supervisor can determine what she has to do to schedule it.

The point of this example is that Inbound EDI transactions are not only about getting data into the database, but about getting it into the database in such a way that it is useful to the operators of the application.

Consider the Exception Cases

An understanding of the process will allow you to see the Exception Cases. For example, if a work order is entered into the system, but it has no bill of materials, the scheduler cannot schedule it. If a work order enters the system, but has the same order number as another that is already in the system, it cannot be entered.

The inbound EDI Transaction is a gateway into the application. Not just any data can come through it. It is never too early to consider situations where it is impossible to enter certain data, and to consider how the EDI Transaction will behave in those conditions.

Parse the Data File by Specifications

Generally, at the beginning of an EDI job, the developer will have access to a sample of the data file, a formal description of the data in that file (the data specifications) and a description of the

EDI transaction as it fits into a workflow process (the job specifications). If you do not have all three of these things, you will need to contact someone involved with the job and get them. This could be your manager, or this could be the customer's project manager.
In general the data specifications will be more specific than the job specifications. The job specifications will often be an email thread, perhaps with attached documents.
Take some time to look at the sample data file. Look at the data specifications. Determine if they match one another. If they do not match, this is the time to straighten it out.
Take some time to study the data specifications too. Often the sample file is a simplification of the specifications. Do not assume that a sample file with one work order means that all data files will have only one work order in the data.

Identify the Data to Import

Once you are sure that the data specifications match the data file, find the specific data fields that you will import into Floormation. In the work order example, this would be the Product Part Number, the Quantity, the Due Date, and any other information that is required.
In many cases, there are several levels to the data. For example, a work order might come with a bill of materials. In this case, there are several database tables that will receive data, and there are relationships between that data. There will be several bill of material rows for each work order row. Each would have a Component Part Number, a Component Description, and a Component Quantity.
Understand that there are many formats that could result in the same data for the database. Continuing the above example, where the EDI Transaction will insert one work order row and many bill of material rows, the data file could have at least two forms. One, all the rows could have the same format, with the work order, product, quantity, and component data on one line for each component. Two, one header row could have work order information, and many component rows could have component data.

Place the Data in the Floormation Schema

When you have identified the data that has to be entered into Floormation, determine where it will go in Floormation, and consider the relationships between the data you have inserted, and between that data and data already in the database.
With the work order, this would mean identifying the table and columns that would store the work data in the work order row, and the table and columns that would store the bill of materials rows. Once you have identified the tables and columns, find out what columns are indexed on those tables. Especially consider unique indexes.
Also consider what additional data will have to go in each row. What status code will the EDI Transaction assign to the work order? The bill of materials rows will have to reference the work order, and that information might not be in the component row of the data file. All of these considerations will help to structure the EDI Transaction program.

Determine the Successful Result

What is inserted into the database, what other changes are made in the database? What fields in the data file contribute to this new data?
An inbound EDI Transaction is usually successful. Between the data file, the database schema, and the job specifications, it should be possible to plan what will happen to the Floormation Database when this successful transaction is complete.

Plan the Error Handling

Since the EDI Transaction is an automated feature, there is no chance to request information from a user in the event of an error. This limitation means that the program will have to be extremely robust and handle any exception case gracefully.

An EDI Inbound transaction usually has to do some checking of existing data before it can insert the new data. In the work order example, this means checking the work order table to see if a work order with the same name exists.
If any of the data validation steps mean that the data cannot be entered into the database, the EDI Program must send a detailed and informative message to a configurable list of recipients. The names on the list can be determined by talking to the project manager for the customer.

Implementation

The Floormation EDI Module uses two files to parse and insert inbound data into the Floormation database: the map file (.sch) and the script file (.jse). When the EDI server reads an inbound data file, it loads the script file. The script file, in turn, calls the map file. The map file parses the data file and loads named fields from it into memory. The script file can then use that data to process the inbound transaction.
By the time you start programming you should have a detailed idea of what the transaction should do, what the end result should be, and what the result of exception cases should be.

Write the Map File

The first step in development is to write a .sch file to parse the data file. The specifications for the parsing language are in another document. This file should parse the sample data file without error. It is important to look at the data specifications, because the sample data file is often the simplest case scenario for the transaction. For example, a work order file sample may contain only one work order, while the specification document indicates that there may be many in the data file. Obviously this has implications for the map file.
In addition, you should ask the project manager for another sample data file, and parse that one too.

Write a Test Driver to Test the Map File

Obviously, there is no way to write a good map file without testing it. It is a good idea, though, to test it in a simple way. Rather than trying to enter the data into Floormation and handle exceptions, you should simply write a driver that sends you an email indicating what values your map file has parsed out of the sample data file.
Later, this test driver may be a good starting point for developing the script file.

Use Design Notes to Design the Script File

This step is where your planning pays off. Since you know what data is in the data file, where it goes in the database, and what you are going to do in case it fails, the logical structure of your script file is nearly complete.
For example, if you know that a work order comes with a bill of materials, you know that for each new work order, you need to check the validity of the work order, insert it, and go into a while loop that inserts components for that work order until there is a new work order. That will be the main structure of the program, and any deviation from that structure will be an exception condition. Every exception condition comes for a reason, and that reason can be emailed to everyone on a mailing list.

Test the Successful Scenario

Tests for successful completion of the transaction should be used to weed out syntax errors and faulty logical constructions. They are the preliminary testing phase for an EDI Script, and passing them in no way makes a program fit for release. When you get the intended result, go on to the next step.

Test Failure Scenarios

This is the real test of an EDI Transaction. Inbound EDI transactions can happen thousands of times a day, for years. This means that there will be some errors in the data coming to them. If bad data gets into the system, there will be a problem, and it will be the worst sort of problem: people will not understand what has gone wrong, and will try to correct it themselves (this means people who know next to nothing), they will inevitably make the situation worse, and there will be an escalation. Managers working for the customer will be called off of their regular duties to straighten things out, and when they find out what went wrong, and they will, they will be angry.

We will look bad.

If bad data prevents data from getting into the system, and no one knows, the problem is not much better: someone will get a telephone call at three-thirty a.m., and will have to log on to the system, pour over log files, find the problem, while half asleep, and report to the caller. This person will not be happy either. The problem is not so severe, because the angry person works for this company, not for the customer.

So, modify the data in Floormation and the data in the test file to test the failure scenarios. Make sure the script file sends an email notification.

Integration Testing

Integration testing should go smoothly if the specifications were accurate and the development was done according to the steps above. Generally, these tests are intended to run through the EDI process on a production machine, testing all of the file transfer mechanisms, directory structures, etc.. These tests are a good chance to run through the process with unknown data and make sure that there are no problems with the final process.

Make Contact and Plan the Schedule

Generally, the customer's project manager will be in charge of the integration tests. They will probably have someone else do the bulk of the testing. You should make contact with this person and find out what they want to test and when.

Learn Production Contacts

The integration test will be a good chance to learn who will be on error notification email lists, and possibly what their job responsibilities are. This is valuable information for the support of EDI scripts. Try to get as much information as possible, and make notes of it for the release document.

Release to Production

Releasing an EDI transaction to a production environment means putting it in the production system and putting control and responsibility for it in the hands of the support staff. They should have as much information as possible about both the process and the transaction.

Install the Program

Take whatever steps are necessary to install the EDI Transaction on the production system. Watch it work fully automatically at least once, and check in on it a few times over the next few days/weeks to make sure it is running correctly.

Release Document

An EDI document should include:

The Name of the Transaction: i.e. Work Order inbound, Materials Request outbound, Production report, etc.

An Overview of the Process: this should include what rows are inserted and updated as a result of the program working, and what error messages are the result of the program handling an exception condition.
A Contact List: that includes the project manager, the customer contacts, and the developer.