Scrittura

Software Version 4.4.10.5

Administration Guide

Document Release Date: October 2021
Software Release Date: October 2021

Legal notices

Copyright notice

The only warranties for products and services of Micro Focus and its affiliates and licensors (“Micro Focus”) are as may be set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Micro Focus shall not be liable for technical or editorial errors or omissions contained herein. The information contained herein is subject to change without notice.

Documentation updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for updated documentation, visit https://www.microfocus.com/documentation/scrittura/.

Support

Visit the MySupport portal to access contact information and details about the products, services, and support that Micro Focus offers.

This portal also provides customer self-solve capabilities. It gives you a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the MySupport portal to:

- Search for knowledge documents of interest
- Access product documentation
- View software vulnerability alerts
- Enter into discussions with other software customers
- Download software patches
- Manage software licenses, downloads, and support contracts
- Submit and track service requests
- Contact customer support
- View information about all services that Support offers

Many areas of the portal require you to sign in. If you need an account, you can create one when prompted to sign in. To learn about the different access levels the portal uses, see the Access Levels descriptions.

About this PDF version of online Help

This document is a PDF version of the online Help.

This PDF file is provided so you can easily print multiple topics or read the online Help.

Because this content was originally created to be viewed as online help in a web browser, some topics may not be formatted properly. Some interactive topics may not be present in this PDF version. Those topics can be successfully printed from within the online Help.

Contents

Chapter 1: Getting Started

This section contains the following topics:

Scrittura Overview

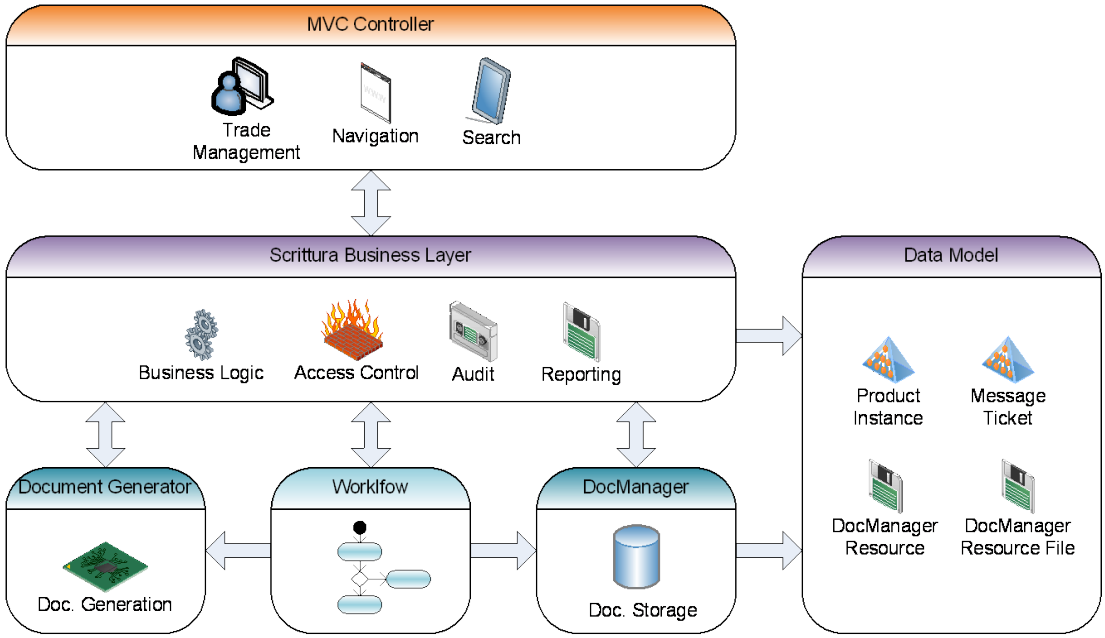
Scrittura is a specialized platform that delivers full automation and control for OTC derivative trade confirmation processing. It mitigates risks by eliminating inconsistencies that would arise from manual processing, while increasing productivity through automation. The Scrittura workflow manages the whole life cycle of trades allowing STP processing (Straight-Through- Processing) of trades or business exceptions to be handled and manually reviewed.

For example, Scrittura:

- Automatically matches trade confirmations with barcodes and document identifiers.
- Automatically matches incoming counterparty documents to outgoing trades.
- Uncovers the conceptual understanding of all data related to trade processing.
- Provides familiar Microsoft Word template management and document creation environment.
- Removes IT bottleneck related to template creation and updates.
- Provides annotation capability to record disputes and offers full audit trail of audit confirmation processing.
- Lets authorized users stamp confirmed trades with electronic signatures.
- Lets users bypass manual review and avoid unintended external edits or version discrepancies that may be overlooked in a manual review.
- Meets the compliance requirements in the derivatives industry.

General Architecture

The following diagram summarizes the key components of Scrittura internal architecture (components are represented with some of their main modules only):



The following components make up the Scrittura core architecture.

- **DocManager** is Scrittura's built-in Document Management System that manages document storage and versioning. DocManager has its own user interface, but is integrated with the Scrittura application.
- **Scrittura Business Layer** handles the business entities and logic in Scrittura. It relies on key components and features such as trade model (Product Instances, Message Tickets), BLogic Rule Engine, Static Data Framework, etc.
- **Scrittura Workflow** defines the sequencing and configuration of the business flow and logic throughout the lifecycle of a trade.
- **Document Generator** is used to generate documents from a set of pre- defined templates as well as live data from the system. In its later version, Document Generator uses the Document Generation Suite (DGS) and is based on Microsoft Word DOCX templates. Templates are designed using Template Manager.

> **NOTE:** Previous solutions (HTML, JSP, and WordML templating) are still supported by Scrittura, although moving to the Document Generation Suite is recommended.

The Scrittura Data Model holds Scrittura database tables required for data persistence throughout the application. For detailed information about these tables and their relationships, see Appendix A: Scrittura Data Model, on page 409.

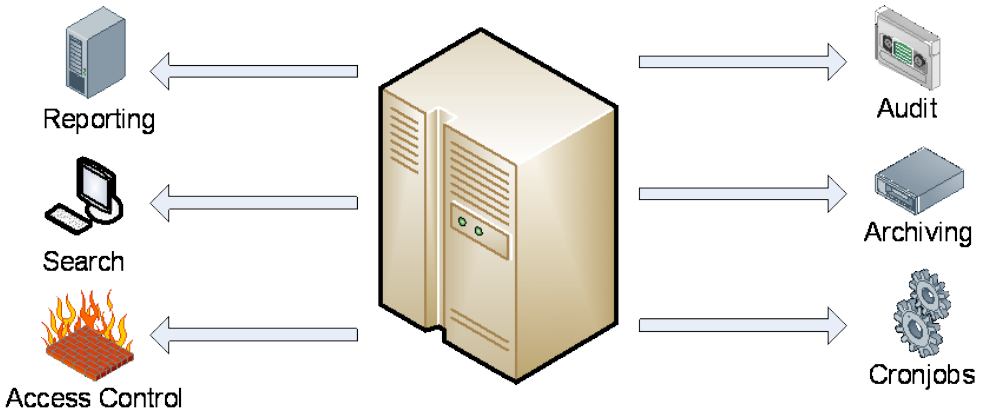
Key Modules and Features

This section provides a quick overview of the native key modules and features of the Scrittura core distribution. They are presented here within the phase in the trade lifecycle where they are most likely to be used (Message Processing, Outbound, Inbound), although they can generally be used throughout the whole lifecycle of the trade. Some features are more application-level capabilities,

and are therefore presented as general features. Full details on all modules, their deployment, and configuration are provided throughout this document.

General Features

The following Scrittura application-level components and features are used not only during the whole lifecycle of a trade but also create a richer user experience (user interface, platform management, and so on).



Audit

Any action in Scrittura–automatic or manual (such as trade move, document creation)–is audited in the system, audit records are persisted in the database.

Access Control

Security is essential in Scrittura and only valid authenticated users can access the application over HTTP or HTTPS connections. Permissions within the application are role-based; different types of access and permissions can be configured within the application for the different roles.

Search

Search criteria can be configured in the application for users to look for specific trades.

Reporting

Scrittura provides a default reporting framework, letting you define on- demand reports and automated reports that can be run at a specific time of the day.

Cronjobs

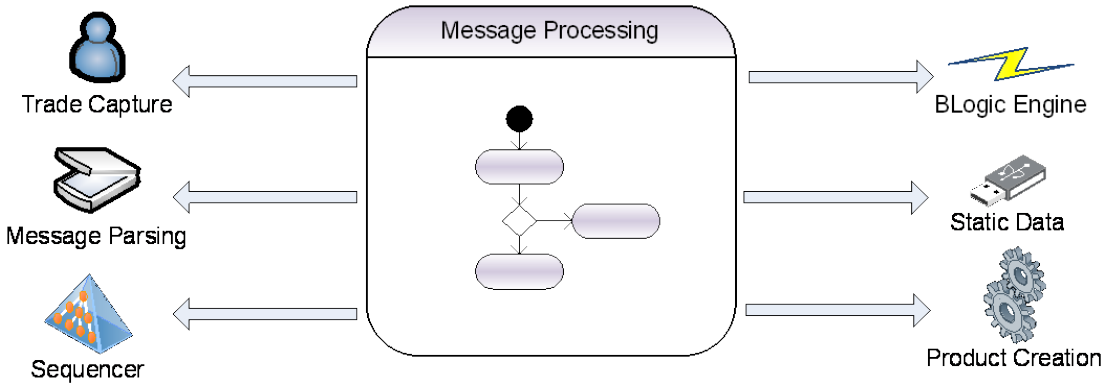
Batch jobs (or cronjobs) can be defined to schedule specific tasks to run (such as reports). Scrittura does not require compulsory housekeeping tasks to be performed on a regular schedule.

Administration

Administration features help the IT team manage the Scrittura platform and its configuration.

Message Processing

The Message Processing phase begins as soon as a trade message reaches Scrittura. Different operations can be performed at this stage, leading to the creation of the actual trade object in Scrittura (Product Instance).



Trade Capture

This module allows the capture of trades from upstream system. Trades are generally sent to Scrittura as text or XML files, via DropBox or over JMS. Scrittura natively supports DropBox mechanisms and listens to a series of incoming JMS queues, the Message Ticket queues.

Message Parsing

Following their capture, incoming messages are parsed for the economic data of the trade to be collected by Scrittura. Although custom parsers can be implemented if need be, Scrittura provides default parser implementations including a generic XML parser.

Sequencer

This module allows the definition of a Sequence in the Message Processing workflow. A Sequence is a section of the workflow where only one single message related to the same trade (or group of trades) can be processed. It guarantees the correct ordering when processing events and market operations related to a trade.

BLogic Engine

Scrittura Business Logic Engine, or BLogic, allows the execution of a list of logic rules. Depending on the rule evaluation result, data derivation can be applied or specific actions triggered. Rules are conveniently entered and organized within Microsoft Excel spreadsheets, where they are executed in the order defined by their priority or from top to bottom when none is specified.

Static Data

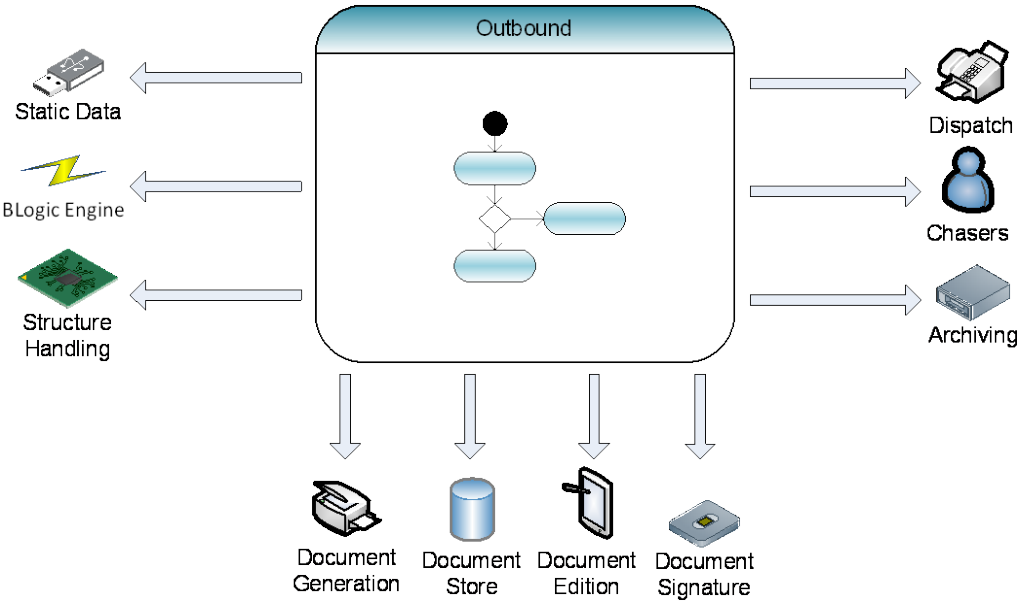
The Static Data Framework allows the configuration, storage, and use of static data in the system (for example, counterparty information). The content of the static data is fully configurable as desired and is persisted in the database. Database tables are dynamically generated at runtime as defined by the configuration.

Product Creation

This is the ultimate step of the Message Processing workflow, where the representation of a trade–the Product Instance–or of a subsequent version is created in Scrittura.

Outbound

Once the trade is created, the trade goes through a series of steps (document generation, review, signature, dispatch, etc.), some manual and some automated.



Document Generation

Document generation (draft and final) is a key component of Scrittura. It allows the generation of documents from pre-defined templates and live system data.

Document Edition

Documents generated using the DGS or WordML can be edited from within Scrittura and their content amended. Changes made to economic data in the edited document can be saved against the corresponding PI. Change tracking can be activated when using the DGS.

Document Storage

Document storage is managed by DocManager, Scrittura's built-in Document Management System, which offers the following possible configurations:

- Filesystem storage
- Database storage
- Storage on an external DMS

Document Signature

Once the confirmation has been reviewed or is marked STP, the confirmation goes through the signature step. Signature can be automated or manual if required. The document signature step is configurable using the available tools (such as, Scrittura signature applet for manual signature).

Dispatch

Once signed, the final confirmation is ready to be dispatched to the counterparty using the preferred medium (fax, email, paper). Scrittura comes with a range of configurable tools to facilitate this step and to integrate with third-party systems.

> **NOTE:** In addition to paper confirmation, trades can be confirmed electronically (such as, through DTCC and ICE).

Chasers

Chasers can be generated and sent to counterparties whose trades are awaiting matching. Chasers can be single or multiple, and generated manually or automatically using Scrittura cronjobs.

Structure Handling

A composite economic model is implemented by default in Scrittura in order to handle structured trades (known as Structures). When used in conjunction with the DGS, confirmations can be generated and managed for the whole Structure, independent components, or groups of components within the Structure.

Static Data

The Static Data Framework allows the configuration, storage, and use of static data in the system (for example, counterparty information). The content of the static data is fully configurable as desired and is persisted in the database. Database tables are dynamically generated at runtime as defined by the configuration.

BLogic Engine

Scrittura Business Logic Engine, or BLogic, allows the execution of a list of logic rules. Depending on the rule evaluation result, data derivation can be applied or specific actions triggered. Rules are conveniently entered and organized within Microsoft Excel spreadsheets, where they are executed in the order defined by their priority or from top to bottom when none is specified.

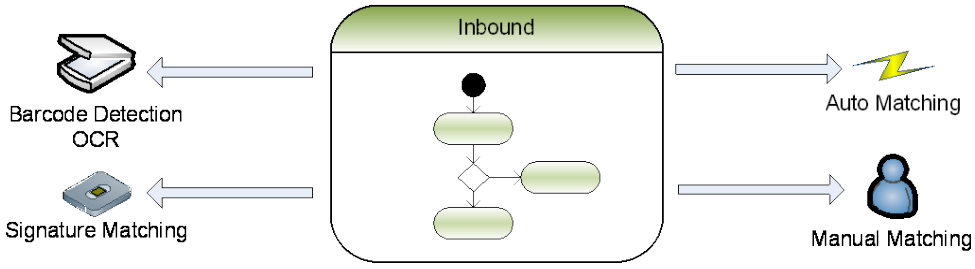
Archiving

Scrittura offers a standard archiving solution that allows archiving matured trades or live trades that have not been accessed for a long time.

Archiving can be automated using Scrittura cronjobs, and archiving criteria is fully configurable.

Inbound

The inbound phase consists in capturing confirmations sent by counterparties and matching them against existing trades in Scrittura.



Barcode Detection and OCR

Barcode detection is a native and configurable feature performed by Scrittura IPS. Advanced OCR (Optical Character Recognition) capabilities are also available when using the OCR Suite of products.

Signature Matching

Signatures of inbound documents can automatically be matched against known signatures. Results are provided with a level of confidence, depending on which document matching can be automated. The use of this capability involves the OCR Suite of products.

Auto-Matching

Matching of inbound confirmations can be automated using barcode detection, OCR, signature matching, or any combination of the three.

Manual Matching

When auto-matching is not possible or not in use, inbound confirmations can be manually matched against the original trade in the system.

Scrittura provides search capabilities on outbound trades and comparison tools in order to perform this task in a user-friendly manner.

Integration and Deployment

Some Scrittura modules are delivered as components external to the core platform (for example, separate servers). Scrittura can also integrate with other servers or with the customer's infrastructure (for example, upstream front-office systems, downstream fax servers, and so on). This section provides an overview of those external components and how they can be integrated to the Scrittura application.

External Scrittura Components

Some services provided by Scrittura, although part of the Scrittura suite, run as separate servers outside the main application.

DocGen Server

Part of DGS, DocGen Server allows the generation of DOCX documents from a set of templates and live data. It communicates with Scrittura over HTTP/S.

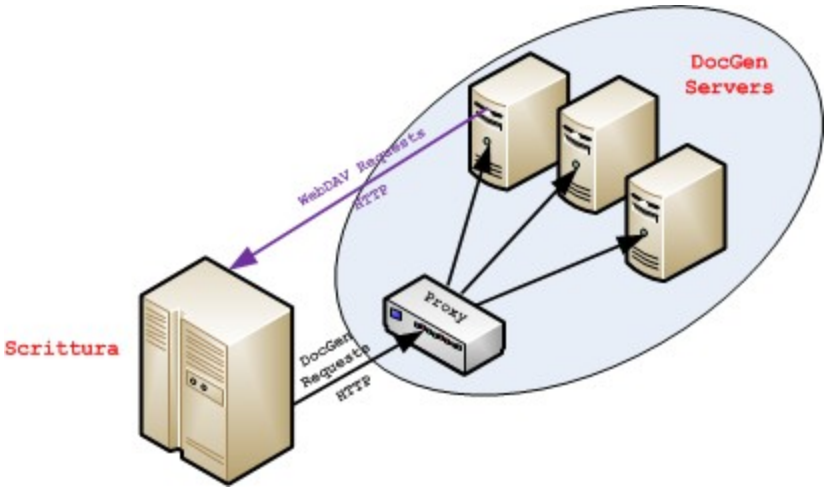
A single instance of DocGen Server can be used to serve requests coming from Scrittura. However, in case of high trade volume, it may be advisable to deploy multiple DocGen Servers and dispatch the requests between the different servers.

Any number of DocGen Servers can be instantiated, on the same or separate boxes. All DocGen Servers must be of the same version for compatibility purposes. A proxy will front the group of servers in order to load-balance the HTTP requests coming from Scrittura.

There are two types of requests to/from DocGen, both over HTTP/S:

- DocGen Requests, initiated by Scrittura in order to generate documents
- WebDAV requests, initiated by DocGen in order to retrieve templates from Scrittura DocManager

The following illustration summarizes the integration of DocGen to Scrittura.



DocGen Servers are standalone stateless applications that only serve HTTP requests, and are therefore not deployed as a cluster. Adequate process monitoring procedures should be put in place to monitor the state of the servers. In case of failure of a server, restarting the failed server is generally sufficient for it to resume its normal operations.

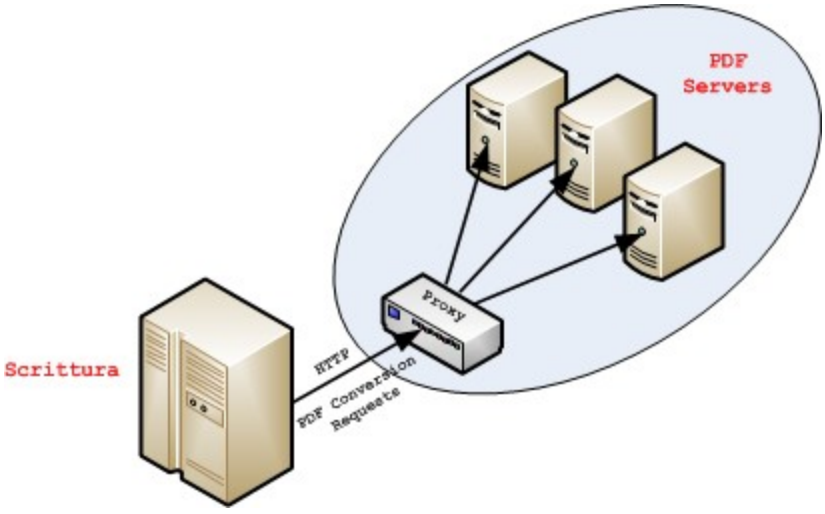
PDF Conversion Server

Part of the DGS, the PDF Conversion Server converts DOCX documents into PDF. It communicates with Scrittura over HTTP/S.

Similarly to DocGen, single or multiple instances of PDF Server can be used. When multiple instances are used, those should run on separate boxes (one server per box) and fronted by a proxy.

Unlike the deployment of a non-clustered Scrittura application, a single WordML-processor should be declared in Scrittura main configuration file, `scrittura-config.xml`, and should point to the proxy itself.

The following illustration summarizes the integration of PDF Server to Scrittura.



PDF Servers are standalone stateless applications, and therefore are not deployed as a cluster. Adequate process monitoring procedures should be put in place to monitor the state of the servers. In case of failure of a server, restarting the failed server is generally sufficient for it to resume its normal operations.

Image Processing Server

The Image processing Server (IPS) is a standalone application that captures inbound documents e.g. from a fax server. It performs essential operations like barcode recognition prior to sending results and documents to Scrittura for further processing and matching.

The IPS takes TIF or image PDF as an input, generally from a scanner or a fax server. It performs image processing like barcode detection prior to sending to Scrittura the document itself as well as an XML message containing the document information (such as, name or barcode content). The XML message is sent over JMS whereas the document is retrieved by Scrittura from a dropbox.

IPS is generally mono-instantiated and communicates with Scrittura by way of dropbox. When setting up the deployment you must make sure that all nodes from the Scrittura cluster have access to the inbound dropbox.

EnConnect

EnConnect is a highly configurable, platform-independent, message- oriented integration server and is part of Scrittura Connectivity Suite. It performs message transformation prior to routing them to their target downstream system.

EnConnect is also mono-instantiated in itself, although each instance may involve multiple Java processes. It communicates with Scrittura via JMS or DropBox. In case the dropbox connectivity is used, you must make sure that only one Scrittura node consumes messages from the DropBox, as detailed elsewhere in this document.

TradeConfirm Server

The TradeConfirm Server is part of the Scrittura Connectivity Suite, although it can be integrated directly with Scrittura. It is dedicated to electronic third-party messaging and is responsible for generating and parsing those messages. The TradeConfirm Server currently supports ICE and certain SWIFT standards, those being available as pluggable libraries, both being subject to specific licensing schemes.

Servers

Scrittura can also be integrated with other servers to enhance Scrittura's native capabilities.

View Server

View Server can be used to convert on-the-fly DOCX documents into HTML so that the documents can be viewed within a browser.

IDOL

Intelligent Data Operation Layer (IDOL) can be used in conjunction with DocManager in its file system configuration to provide advanced search features on the document base.

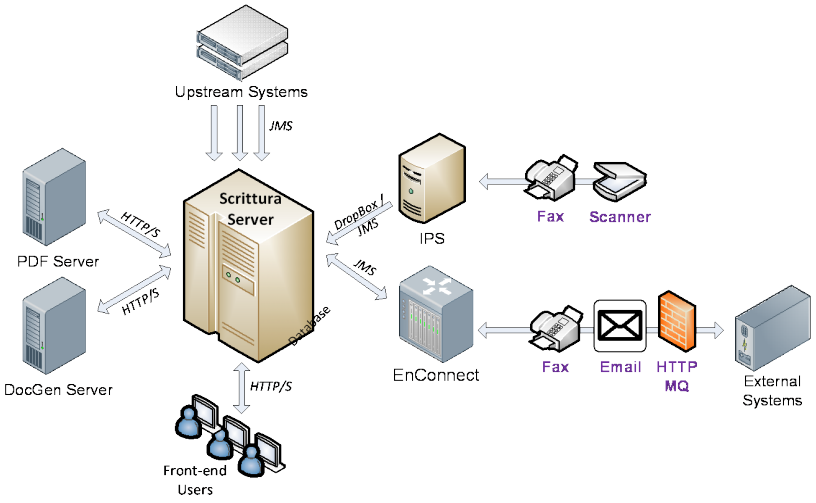
Teleform and Image Server

The Teleform and Image servers provide advanced features such as Optical Character Recognition (OCR) and signature matching.

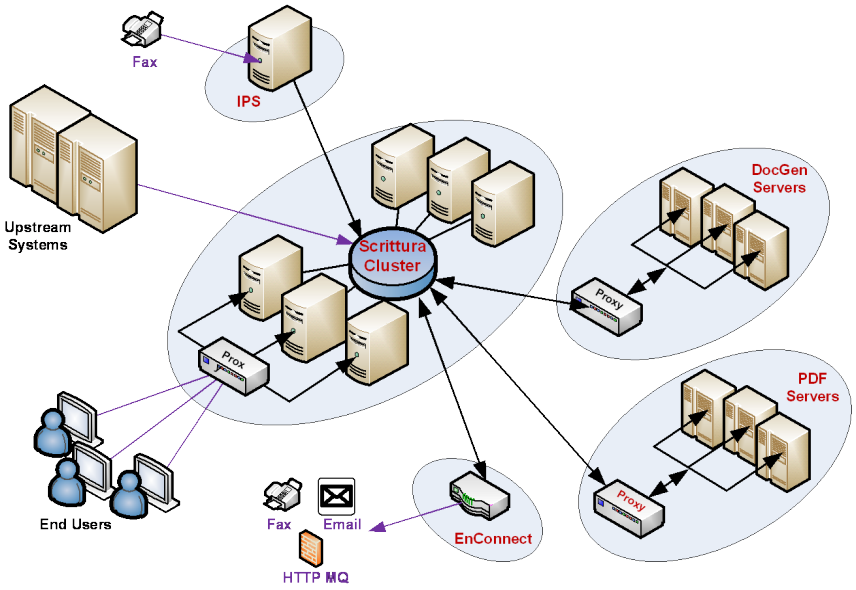
Deployment Diagrams

The diagrams in this section depict example Scrittura deployments to highlight how the components and systems interact.

In a standalone deployment, Scrittura runs as a single server instance.



In a clustered deployment for large scale systems, multiple Scrittura nodes work collaboratively in a cluster. In this example, the DocGen Server and the PDF server are also multi-instantiated and run outside of the Scrittura cluster.



Implementation Guidance

When implementing Scrittura, most tasks can be completed in parallel and are iterative. Use the following information as guidance for completing your implementation. The remainder of this guide provides full details on how to configure and customize Scrittura during implementation.

The following are the essential core Scrittura configuration and customization tasks.

- **Data Dictionary Definition**

  The first essential step is to define which data will be held by trades, in other terms, your Data Dictionary. Once defined by the business team, the Data Dictionary is translated into its Scrittura equivalent, the Product Definition.

- **DocManager Configuration and Customization**

  DocManager hierarchy and meta-data are defined by way of configuration. If an external DMS is used to store documents, the corresponding connector should be developed in order for DocManager to interact with this external DMS.

- **Workflow Creation**

  The workflow defines the processing performed on trades throughout their lifecycle. This step includes configuration using the Workflow Modeler and Scrittura configuration to add the new workflow. This typically involves the creation of the trade pre-processing workflow (MessageProcessing), outbound workflows, and inbound workflow. At this stage, any custom logic should be implemented or business rules configured. The confirmation medium (paper or electronic) should also be identified at this stage.

- **Global Scrittura Configuration**

  Global Scrittura configuration involves defining permissions, audit details, and search criteria users will have access to. User permissions are essential to control user access to the queues in the system. Permissions also define which documents or queues and trades are visible to the users.

- **User Interface Configuration and Customization**

  A large part of the user interface can be configured directly in the Scrittura configuration (queue views, trade detail screens, and so on). If needed, custom pages can also be added to the Scrittura application. The various features offered to end users are defined at this stage.

In addition to the core configurations, the following tasks are performed during implementation.

- **Template Design**

  Scrittura offers a powerful document generation solution based on Microsoft Word DOCX templates. Documents are generated using these templates and trade data from the live system.

- **Report Customization**

  Custom reports can be defined using standard solutions like BIRT or Jasper Reports. Once reports are designed, they are integrated into the Scrittura configuration.

- **Administration Tools Customization**

  Scrittura includes a series of tools for administration purposes. Among others, housekeeping tasks can be implemented using Scrittura cronjobs and the archiving process can be customized as appropriate for your environment.

- **Integration with Other Systems**

  Scrittura integrates with upstream and downstream systems in order to perform its duties. Upstream systems are generally front-office systems while downstream systems can be fax,

email server, or electronic messaging systems such as, DTCC, ICE or SWIFT. Connectivity to these systems must be established.

For new Scrittura implementations, you should first deploy the out-of-the-box default Scrittura client in order to better understand the concepts developed in this guide.

Chapter 2: Scrittura Configuration

This section provides an overview of the different files and folders involved in the configuration of the Scrittura application. Also included are the details of one of the main configuration files, scrittura-config.xml.

This section contains the following topics:

Scrittura Configuration Overview

This section provides a general overview as to how the Scrittura configuration is organized and which files or folders are involved in the configuration.

Scrittura Configuration Files

Scrittura uses XML or properties files for its configuration. These configuration files, all located under the /config folder of the live repository, let you customize the application without changing the core functionality.

The following table lists the Scrittura configuration files for the different modules available in the distribution, with a reference to the location in this guide where full explanations can be found.

File Name	Description	Reference
`scrittura-config.xml`	Scrittura core configuration files.	Scrittura Configuration, above

`docmgr-config.xml`
`entity-types.xml`
`docmgr-external- config.xml`
	DocManager configuration files, including the configuration of the hierarchy model and integration with an external Document Management System.	DocManager Configuration and Administration, on page 122

`workflow.xml`	Workflow engine configuration file.	Workflow Configuration, on page 73
`scheduler.xml`	Scheduling of Jobs.	Scrittura Utility Modules, on page 267: Job Scheduler Configuration, on page 267 and Archiving, on page 274
`signatures.xml`	Configuration of applicable Signatories and signature	Outbound Workflows, on page 296

`signature-config.xml`	process.	
`inbound-config.xml`	Inbound module configuration.	Inbound Workflow, on page 309

`log.cfg`
`encryption-key.ky`
`startup-config.xml`
`custom.properties trade-simulation- config.xml`
	Scrittura run-time administration files:
	• Log configuration
	• Password encryption key
	• Startup configuration
	• Optional run-time properties
	• Trade simulation (for development and testing only)
		Scrittura Administration and Run-Time, on page 369

`roles`	Scrittura Roles configuration file.	Scrittura Administration and Run-Time, on page 369
`matching.dict`	Matching dictionary for Inbound module.	Inbound Workflow, on page 309
`mime.types` `mime.icons`	MIME types configuration files.	DocManager Configuration and Administration, on page 122
`adg.conf`	V2 template parameter configuration file	See previous versions of the documentation

`mvc-extend.cfg`
`scrittura-queue-view-config.xml`
`economic-panel- config.xml`
`general-ui-config.xml`
	User interface configuration:
	• Custom events
	• Bulk screens
	• Trade detail screen economic panels
	• General user interface configurability
		User Interface Configuration, on page 157

`birt-report-config.xml`	BIRT reporting configuration	Search and Reporting, on page 220
`blogic- factory.properties`	BLogic configuration	BLogic Business Engine, on page 203
`docgen-config.xml` `confirmation-group-`	DOCX document generation configuration	Outbound Workflows, on page 296

`mapping.xml`		
`datamapping-config.xml` `datamapping-ui-config.xml`	Static Data Framework data model and user interface configuration	Static Data Framework, on page 238
`email-dispatch-config.xml`	Email dispatch configuration	Outbound Workflows, on page 296
`sequencer.properties`	Sequencer configuration	Message Processing Workflow, on page 281
`spring-config-http.xml`	Spring services configuration	Outbound Workflows, on page 296
`structured-product-config.xml`	Structured Product configuration	Structured Products, on page 352

NOTE: All configuration files are located in the Scrittura \config folder of the live folder, except custom.properties which is located directly under the live folder.

Configuration Folders

Core and optional Scrittura modules may include additional folders as part of their configuration. Those folders contain a series of business configuration files for that module (such as, rule files, templates) that are loaded by Scrittura along with the module base configuration files.

The following table lists the folders used by Scrittura, and modules they apply to with a reference to the location in this guide where full explanations can be found.

Folder	Description	Reference
`\birt`	BIRTreport templates	Search and Reporting, on page 220
`\blogic`	BLogic rule files	BLogic Business Engine, on page 203
`\DGSTemplates` `\productviews`	DOCX templates and product views to use with Document Generation Suite (DGS)	Outbound Workflows, on page 296
`\emailTemplates`	Email templates	Outbound Workflows, on page 296
`\products`	Scrittura Product Definitions	Product Definitions, on page 55
`\signatures`	Signature images	Outbound Workflows, on page 296

`\templates`	BeanShell scripts and HTML templates	Outbound Workflows, on page 296
`\xmlParserConfig`	CSV configurations used by the generic XML parser	Message Processing Workflow, on page 281
`\workflow`	Scrittura workflows	Workflow Configuration, on page 73

NOTE: Configuration folders are located directly under the live folder.

Mandatory and Optional Configurations

A part of the configuration is mandatory for Scrittura to be able to run, whereas other modules may be optional.

For example, Scrittura requires at least one workflow, one Product Definition, and that DocManager is configured. On the other hand, the use of some modules, such as the “BIRT” module, is optional and would depend on the business requirements.

Optional modules are listed in the Scrittura startup configuration defined in `startup-config.xml`. If not required, those modules can be disabled. For full details, see Scrittura Administration and Run-Time, on page 369.

Configuration File Validation

When loaded by Scrittura, XML configuration files are validated against a DTD or XSD. DTDs are located under the /dtd folder of the live repository; they are copied there from the Scrittura distribution during the deployment process.

The following DTDs and XSDs are used to validate the different configuration files.

Configuration File	DTD or XSD
birt-report-config.xml	birt-report-config.dtd
confirmation-group-mapping.xml	confirmation-group-mapping.dtd
datamapping-config.xml	datamapping-config.dtd
datamapping-ui-config.xml	datamapping-ui-config.dtd
docgen-config.xml	docgen-config.dtd
docmgr-config.xml	dms-config1_0.dtd
docmgr-external-config.xml	spring-beans-2.0.xsd spring-aop-2.0.xsd
economic-panel-config.xml	economic-panels-config.dtd

email-dispatch-config.xml	email-dispatch-config.dtd
entity-types.xml	etype1_0.dtd
general-ui-config.xml	general-ui-config.dtd
inbound-config.xml	inbound-config1_0.dtd
scheduler.xml	scheduler.dtd
scheduler.dtd	scrittura-config.dtd
scrittura-queue-view- config.xml	scrittura-queue-view- config.dtd
signatures.xml	signatures.dtd
signature-config.xml	signature-config.dtd
spring-config-http.xml	spring-beans.xsd
startup-config.xml	startup-config.dtd
structured-product- config.xml	structured-product- config.dtd
trade-simulation-config.xml	trade-simulation-config.dtd
TransformVariableConfig.xml	TransformVariableConfig.dtd
workflow.xml	workflows.dtd
Product Definition files	proddef.dtd
XML Workflow files	xpdl.dtd
environment.xml (BLogic)	blogic-env.dtd

Reloading the Configuration

In order to prevent any uncontrolled change in the live configuration that would be caused by human error, Scrittura does not load its configuration directly from the filesystem upon restart but from the database. Therefore, the Scrittura configuration must be reloaded into the database following any configuration change so that the live system can subsequently reload it.

A special Scrittura process fulfills this purpose, the SetConfig process. This process loads the configuration from the filesystem to the database and to the live system. The SetConfig process must be run after any change made to the configuration files in order to apply those changes to the live system.

For full details, see Scrittura Administration and Run-Time, on page 369.

Scrittura-config.xml File

The `scrittura-config.xml` file, located in the \config folder of the Scrittura live repository, controls major aspects of Scrittura. In addition to defining general settings in the application, it links together

the core modules whose configurations are mandatory in Scrittura. These core modules include, but are not limited to the following.

- Scrittura workflows to be used in the system
- Workflow queue organization and access
- DocManager hierarchy and mapping
- Audit threads
- Public searches, filters, and search variables
- Archiving

The following node elements can be defined in the scrittura-config.xml file.

<scrittura-config> Node

The `<scrittura-config>` node is the root node of the `scrittura- config.xml` file and has the following attributes.

Attribute	Required/ Optional	Description
CommonReferenceID-field	Optional	Defines the field in the entity model to be used in order to keep track of the `CommonReferenceID` of a trade that a DocManager document references.
database	Required	Defines the database type used by Scrittura. Possible values: `Sybase, MSSQL, or Oracle`
date-format	Required	Defines the display format for dates.
date-parse-format	Required	Defines the default user interface input format for dates.
decimal	Required	Defines the character used as the decimal separator for Currency fields.

docmgr-path	Required	Default DocManager folder structure for the Product Instances (PI), which can be overridden in the Product Definitions.
		This path must correspond with the entity type model as defined in the file `entity-types.xml`. The PI folder can be created either by being attached to the DocManager Library Root or by being hidden as a rootless folder. Defining the path with or without a "\" at the beginning determines the behavior when creating the PI folder. For example:
		Rootless folder:
		`docmgr-path=Counterparty\Product\Deal`
		Root folder:
		`docmgr-path=\Counterparty\Product\Deal`

document-version-field	Required	Defines the field in the entity-model to be used as a tracker that keeps track of which version of a trade a DocManager document references. The Scrittura system uses this information to determine when to create new versions of the document.
enable-refresh	Required	`True/false` indicator that lets you ask for a refresh every 5 seconds of your present queue list, by clicking on the circle of arrows next to the queue list title. When you leave the queue list, the refresh feature shuts off until it is requested. Each request performs a potentially expensive database query, so the default behavior is to keep this refresh off. The `refresh-secs` attribute determines the number of seconds between refresh cycles.

fa-default-length	Optional	Default character length for each of the variables in FA and Archiving Tables. This default length is used for the FA table only, and is not applied to the actual length of the variable in the BLOB. For example, if `fa-default-length=50`, and the variable does not have a max-length defined, a value of 70 characters long will be stored fully as 70 characters in the BLOB, but only the first 50 characters are stored in the FA table. Hence, only the first 50 characters can be used in the search term. Format: positive integer; `50` if not defined
fa-table-column-size	Optional	Specifies the column size that holds the variable names in FA and Archiving Tables. The default is set to 30 characters.
history-view	Required	Defines the view file for inclusion in the audit/history page.
host-name	Optional	Scrittura server hostname.
local-port	Optional	Scrittura server HTTP port.
lock-expiration-secs	Optional	Defines the number of seconds after which a user's lock on an activity expires. This prevents users from holding locks for days (for example, if a user exits the browser without closing a view). Default: `43200` seconds (12 hours)
log-session-locks	Optional	Boolean value specifying whether the debug information about session or session object locking should be logged. The session or session object locking is used in the MVC Controller and Scrittura tags to prevent a duplicate submission of the form. Possible values: `true` or `false`; `false` if not defined
logic-folder	Required	Location of the BeanShell scripts
max-items-in-cache	Optional	Integer defining the maximum size of the Scrittura cache. Default: `100`
message-process message-activity	Optional	Default process and activity to be used as a start point for workflow items, unless overridden in the Product Definition.

monitor-ext	Optional	The extension (case sensitive) of the files to pick up from the monitor folders. Files with any other extension not defined in this attribute are ignored by the import process. Leaving this attribute blank results in every file in the monitor folders being picked up. The files themselves may be in XML format or Scrittura tag file format, but they must use a common extension, such as `.IN`. For example: `monitor-ext=.IN` When designing an interface to put files in the monitored folder(s), complete in the following order to avoid errors with partial file writes: 1. Create the new tag file with a temporary extension - not the configured monitor-ext value above. 2. Write the file. 3. Close the file. 4. Rename the file to use the monitor-ext extension specified above.
monitor-file-access	Optional	Specify the locking mechanism to handle concurrent accesses to the dropbox. It can take one of the following three values: • `read`. Files waiting in the dropbox are read then deleted by the monitor, which does not apply any locking mechanism. This is the default option and the fastest. For this option to work correctly, files must be written in the dropbox in an atomic manner, such as, by a “move” operation rather than a “copy”. • `lock`. This option should be selected when Scrittura runs in a cluster where multiple nodes access the same dropbox. Prior to reading a file from the dropbox, the monitor attempts to create a .lck file and only consumes the file when this creation is successful. Then similarly to the read option, the dropbox file is read and deleted along with its `.lck` file once complete. This prevents two nodes from reading the same file from the dropbox and creating duplicate trades in the system. • `exclusive`. In addition to creating a .lck file as for the lock option, the monitor attempts to take an exclusive lock on the dropbox file itself and only consumes the file if both operations are successful.

		The purpose is to prevent the monitor from consuming a partial file when a “slow” external file publisher is used. Note that the file publisher must also take an exclusive lock for this option to work. Operating System constraints and limitations may also apply.
monitor-folders	Optional	Dropbox directory paths that Scrittura monitors for message files being dropped in for processing. The monitor waits ten seconds between checks for additional dropped files. If multiple directories are listed, they must be separated with space characters. Also, spaces inside directory names are not supported. Using this technique for dropping in messages (as opposed to an MQ Series or JMS queue) requires the inclusion of the startup class com.ipicorp.scrittura.util.LoadTagThread . See the startup-classes attribute. The Trade Monitor view specifically monitors the activity of this class: `http://[machinename]:[port]/scrittura/jsp/importstatus.jsp` This view appears by default in the top menu and is by default accessible only to users in the 'admins' role. It shows the last time the directory was checked for dropped files and allows a forced manual restart. Classes for interpreting the dropped files must be configured in the `<message-type>` attribute. (These classes must also be specified for JMS and MQ Series queues.) The following is an example of monitoring two directories for dropped files. Note the use of forward slashes: `monitor-folders="c:/temp/scritturadrop //netmachine/f$/temp/scritturadrop"`
record-size	Optional	Maximum width of a record in FA tables. SQL scripts that create FA tables are generated in a way that the number of bytes in each record is not over this value. If this attribute is not defined, the default value used depends on the type of database: `1908` for Sybase, `65535` for Oracle, and `8060` for Microsoft SQL Server. Format: positive integer
refresh-secs	Optional	Defines the number of seconds between refresh cycles when `enable-refresh` is set to `true`.
scratchpad	Optional	Temporary folder used along with `scratchpad-template`.

scratchpad-template	Optional	Location of the WordML or JSP base templates for document generation.
separator	Required	Defines the character used as the thousands separator for Currency.
simfiles-folder	Required	Defines the location of tag files for trade.
ssl-port	Optional	Scrittura server SSL port.
template-folder	Required	Defines the location of the base HTML templates for document generation.
templates-in-dms	Optional	When set to true, Scrittura checks for customized `JSP` files (for example, templates) in the database first. If it does not find any, it uses files from the `EAR` file. When set to false, Scrittura uses files from the `EAR` file. Possible values: `true or false; false` if not defined
template-recheck-sec	Optional	Number of seconds to wait before checking for updated pages in DocManager. This value can be changed in the SetConfig screen. Default = `10` seconds.

trade-throttle-secs	Optional	The amount of time, in seconds to throttle message delivery, whose default value is 10 seconds.
		
		The throttles are linked to up to five configurable Scrittura ticket listener MDBs. Each of these MDBs can listen to a JMS queue (for example, `scrittura_tickets`). For each of these additional JMS queues, you can set up a separate throttle (`trade-throttle-secs2` to `trade-throttle- secs5`).
		
		Any value from 0 to the configured JTA timeout value on the J2EE container is valid. Values greater than the JTA timeout value will work but will generate a `TransactionTimeoutExceptions` entry in the log.
		
		This parameter controls the rate at which inbound JMS messages are accepted. At most, one message is accepted every trade-throttle-secs. For example, if the value is 10 and three messages arrive simultaneously at time T, all three are delivered to the JMS queue immediately but will be processed (that is, sent into the workflow) at:
		
		`T`
		
		`T+10`
		
		`T+20`

		If at `T+15`, two more messages arrive, they are processed at:
		`T+30`
		`T+40`
		If two more messages arrive at `T+53` (13 secs after the last message), they are processed at:
		`T+53`
		`T+63`
		In general, for best performance, configure this value to represent the average amount of time an item takes from drop-in until it reaches its first queue. This prevents the workflow engine from spending all its time processing new messages. Adjustments to this value do not substantially improve actual performance of the workflow engine (in terms of actions per second). Rather, performance improvement may be perceived by making adjustments that control the rate at which the engine processes new messages (a process that is typically more processor-intensive than user activity on existing product instances).

use-oracle-rule-optimizer	Optional	Provides control over whether to use cost-based or rule-based optimization when using ORACLE. This controls whether the generated SQL includes the `/*+ RULE*/` optimizer hint. It affects what optimization strategy Oracle uses to construct its query. The default remains backwards compatible. Possible values: `true or false` Default: `true`
use-upper-searches	Optional	Boolean flag to be set to true in order to perform case-insensitive Scrittura trade searches (quick search and advanced search). Default: `false` **NOTE:** This flag has no effect on DocManager and Static Data Framework searches, which are case-insensitive.
use-upper	Optional	Boolean flag to be set to true in order to use case-insensitive role conditions. Default: `false`
user-preference-manager	Optional	Name of the class used to determine user preference (date format and currency format) behavior, by default: `com.ipicorp.scrittura.util.IpiUserPreferenceManager`

v2-font-color	Optional	Sets the font color for variables modified by users when they appear in documents generated from HTML templates. This is opposed to variables in JSP templates, which take on the font color specified in the style sheet.
v2-sys-font-color	Optional	Color used to indicate system changes to variables before passing them to the version 2 HTML-based document generation engine. Its values are the standard color names (such as red, blue, and so on).

<wordml-processor> Node

The `<wordml-processor>` node can be multi-instantiated and allows the declaration of the PDF Conversion Servers (also called WordML Processors) to be used for converting Microsoft Word confirmations (DOC or DOCX) to PDF.

The `<wordml-processor>` node has the following attributes.

Attribute	Required/ Optional	Description
hostname	Required	Hostname of the PDF server to use.
port	Required	Port of the PDF server to use
ssl	Optional	Boolean flag that, when set to true, allows the use of HTTPS.

PDF conversion may be a lengthy step in the Scrittura workflow, depending on the number of pages the document may contain. Although the bulk of the PDF processing takes place outside Scrittura on a separate server, items in the workflow have to wait for its completion prior to moving and sometimes have to queue for previous items to be processed.

To prevent potential workflow bottlenecks, multiple PDF servers can be defined, one `per <wordml-processor>` node. The load is then dispatched by Scrittura between the different PDF servers available.

<product-def> Node

The `<product-def>` node contains the name of an XML Product Definition document stored in DocManager in the Product Definitions folder.

Only the Product Definitions listed in such a `<product-def>` node are recognized by the application when a `SetConfig` operation is performed.

`<product-def>` does not have any attributes; its body contains the name of the XML Product Definition file, located under the `\products` repository.

Example

```
<product-def>proddef-fxo.xml</product-def>
```

<dms-field-ref> Node

The `<dms-field-ref>` node defines the mapping between the indexes and fields used by the DocManager entity types (as defined in `entity- types.xml`) and the Scrittura Product Instance variables.

The `<dms-field-ref>` node has the following attributes.

Attribute	Required/ Optional	Description
name	Required	Name of the DocManager field or index.
ref	Required	Scrittura variable to substitute for this value.

The mapping defined by `<dms-field-ref>` can be overridden at the product level by adding the superseding mapping directly in the corresponding Product Definition. For full details, see Product Definitions, on page 55.

<global-role> Node

Global roles act as keys to data. Users have access to all the data for which they have a key.

Queue roles act as filters. These roles also apply to search conditions, search- based reports, and document generation.

Note the following:

- Global roles may be overridden in a queue.
- If any role in the queue requests an override, that override applies only to a global condition for the same role name.
- To override ALL global conditions, you must have an override role for each global condition.
- There are override flags on the queue that will ignore global roles for this queue.
- You cannot override the global roles in search results.

Queue roles, global roles and conditions are unioned, meaning that you can see an item if any condition allows it. The results represent the intersection of the two sets. You can see all the items for which any queue-specific role condition grants you access AND any global role condition grants you access. However, if you are granted access through a role condition but not a global condition (or vice versa), you do not have access.

Conditions can be defined in this element, but require SQL syntax as opposed to standard Java syntax, as in the following examples:

Example: `Equal`

```
<global-role name="signers_a">
<condition>TradeType = 'VCO'</condition>
</global-role>
```

Example: `Not Equal`

```
<global-role name="signers_a">
<condition>TradeType <> 'VCO'</condition>
</global-role>
```

Example: `Joins`

```
<global-role name="signers_a">
<condition>TradeType = 'VCO' AND Currency = 'USD'</condition>
</global-role>
```

Both `<queue>` and `<role><condition>` nodes have a corresponding optional `true/false` attribute, `override-global`.

Example 1:

```
<queue name="" activity="" override-global="true">
```

Example 2:

```
<role name="signers_a">
<condition override-global="true"> TradeType='NDF'
</condition>
...
</role>
```

<message-type> Node

The `<message-type>` node is used to specify the Message Parsing classes, used for parsing different formats of input messages. Scrittura must parse inbound messages into value pair maps before it can use them.

Inbound messages can be delivered through JMS or MQ Series queues or they can be dropped into the dropbox (an attribute of `<scrittura-config>`).

The `<message-type>` node has the following attributes.

Attribute	Required/ Optional	Description
name	Required	Name of the parser.
class	Required	Parser processing class.

Scrittura natively provides the following parsers and also provides the ability to define custom parsers.

Message Type	Class
Flat XML	com.ipicorp.scrittura.messages.FlatXmlParser

Generic XML	com.ipicorp.scrittura.messages .GenericXmlParser
Tagfile	com.ipicorp.scrittura.messages .TagMessageParser

When parsing a message, Scrittura attempts to use the parsers in the order defined in scrittura-config.xml and will return a successful result as soon as it finds an applicable parser.

- **Generic XML Parser**. Scrittura's Generic XML Parser allows the parsing of XML messages that follows any schema. It is based on XPath-like expressions, variable mapping being defined in configurable CSV files. It comes in two flavors, one for standalone trades, and one for Structured Products.

  For full details, see Message Processing Workflow, on page 281.

- **Tagfile Parser**. The Tag format follows a simple "variable value pair" structure, the variable name being delimited with curly braces:

  ```
  {CommonReferenceID}=123

  {ProductDefID}=swap

  {Countarparty}=Bank of the North

  {Currency[A]}=USD

  {Currency[B]}=JPY

  {SettlementInstructions}=Line 1 of instructions line two of instructions
  ```

- **Custom Parsers**.Although Scrittura natively offers a series of parsers to parse the most common message format, it is possible to define custom parsers and add them to the list of parsers in scrittura-config.xml as `<message-type>` nodes.

  Custom parsers must extend the MessageParser interface located in the com.ipicorp.scrittura.remote package and implement the following method, whose argument is the content of the message to parse.

  ```
  public Map parseMessage(String msg) throws MessageException;
  ```

  This method should return the map of variables/values retrieved from the message, or null if the message format is not the one expected by the parser.

<image-processing> Node

If Scrittura receives image processing results through JMS, you must specify the class or classes that receive and process the results within Scrittura.

This is accomplished by adding <processor> nodes as child nodes of `<image-processing>`. A `<processor>` node has the following attributes.

Attribute	Required/ Optional	Description
name	Required	Name of the image processor class. If the JMS header matches this name, then this processor is used for processing the image

		processing results.
class	Required	Class for this image processor. The default class for receiving and processing JMS image processing results is the following: `<processor name="imageprocessing" class="com.ipicorp.scrittura.imagep rocessing.JMSInboundImageProcessor" />`

<menu> Node

The `<menu>` node defines the organization, visibility, and access to Scrittura workflow queues from within the web application.

`<menu>` has a single required attribute, name, which is used to assign a name to this menu. `<menu>` supports any number of `<queue>` nodes as child nodes.

A `<queue>` node is used to define a business queue within the application. It maps to one or multiple workflow manual queues and views, and access to this queue can be defined there. A `<queue>` child node has the following attributes.

Attribute	Required/ Optional	Description
name	Required	Name of the queue as it will be known by the Scrittura platform.

activity	Required	Fully qualified name of the main workflow activity this queue maps to, using the following syntax:
		`[ProcessID].[ActivityID]`
		If the workflow process ID or activity ID has been defined to contain space characters, those should be replaced by an underscore.
		Example
		`standardOutbound.Pending_Review`
		When an activity is set to "empty", a blank row displays in the queue list (or the display name displays, centered).
		NOTE: It is possible to add secondary queues.

override-global	Optional	Allows overriding global roles in order to specify queue- specific roles.

Example

```
<menu name="Outbound">
<queue name="Pending Review" activity="Outbound.Pending_Review">
…
</queue>
…
```

```
</menu>
```

A `<queue>` node can have multiple child nodes that allow the definition of the following.

- Columns that will display for this queue
- JSP views that will be used for the queue trade list and individual trades
- Roles to define user access to the queue and conditional role-based views
- Additional activities to include in that queue

Queue Columns

Queue columns are defined child nodes of the <queue> node as one of the following.

- A `<column-set>` node, in order to define the whole set of columns at once
- Individual `<column>` nodes, one per column to display

`<column-set>` has the following attribute.

Attribute	Required/ Optional	Description
name	Required	Name of the column set to use for this queue. Column Sets are defined in another part of `scrittura-config.xml`, as `<column-set>` nodes. For more information, see <column-set> Node, on page 48

`<column>` has the following attributes.

Attribute	Required/ Optional	Description
display	Optional	Column header. To force a carriage return (such as ` ` tag) in this header, use the HTML codes `&lt` and `&gt` within this display variable. For example, with a value of `"Document Specialist"`, the header will be displayed on two separate lines, as follows: `Document` `Specialist`
variable	Required	The Product Instance variable to display.

NOTE: It is possible to mix <column-set> and <column> nodes, when additional columns are required.

Views

Views are defined using the <view> child node of the <queue> node and have the following attributes.

Attribute	Required/ Optional	Description
name	Required	Name of the view element.
type	Required	Type of the view element, either bulk (bulk screens), edit (data enhancement), view (property sheet), or custom.
view	Required	URL of the pane that displays the actual trade details (left side of the page).
frameset	Optional	Frameset to use for the trade detail screen.
script	Optional	BeanShell script, run when the view is closed.
show-on-search	Optional	Boolean option that specifies whether to offer this view as a link in the bulk screens.
validator-class	Optional	Name of the class responsible for validating any data saved from the view.

Roles

User access to the different queues is based on their role. It is possible to define which roles will have access with which permissions (read or write). Conditional views based on the user role can also be defined.

Role-based restrictions for a queue are defined using the `<role>` child nodes of the `<queue>` node for each queue. If none is present, the queue is open to all users as read-only. If any roles are present then the queue is restricted to the specified users with the access rights. To be recognized by Scrittura, roles must be defined as groups in the application server security realm and listed in the roles configuration file.

A `<role>` child node has the following attributes.

Attribute	Required/ Optional	Description
name	Required	Name of the Scrittura role.
access	Optional	Defines the access rights to a queue for users in that role. Possible values are `read` or `write`. The default access rights are set to read only. The write permission also provides read access. If users are members of more than one role, they are granted write access in an additive manner, such that if they are in any role with write access, they have write access.

Role-based views can be defined by adding `<view>` tags as child nodes of the `<role>` node.

Example: users in the Drafters role (and only them) will see the bulk view specified by the nested <view> tag

```
<role name="Drafters" access="write">

<view name="Review"

type="view" view="/jsp/bulkBase.jsp?panels=next" />

</role>
```

Conditions can also be added would the access permissions need to be refined. A condition is specified as a `<condition>` node, child of the <role> node, whose body is a pseudo-SQL conditional expression that determines which items this applies to. The expression must be wrapped with `CDATA`.

```
Example: users need to belong to the Drafters role in order to see the bulk view
specified by the nested <view> tag, but will only have access to trades whose
variable drafterUser1 matches their user principal:

<role name="Drafters" access="write">

<condition><![CDATA[drafterUser1='USERID']]></condition>

<view name="Review"

type="view" view="/jsp/bulkBase.jsp?panels=next" />

</role>
```

Following are pseudo SQL examples:

```
<condition><![CDATA[Generate_As_SWIFT<>'YES']]></condition>

<condition>

<![CDATA[(Generate_As_SWIFT<>'YES'

AND Front_Office_System='IRP' AND (Portfolio_Group_Name='WTTT' OR Portfolio_Group_
Name='WTTF'

OR Portfolio_Group_Name='WTSO'))]]>

</condition>

The system supports only one condition per role. That means that instead of a
configuration such as:

<condition><![CDATA[TradeType='Swap']]></condition>

<condition><![CDATA[Currency='USD']]></condition>
```

you must define a single condition such as:

```
<condition>

<![CDATA[(TradeType='Swap' AND Currency="USD")]]>

</condition>
```

Additional Activities

It is possible (though optional) to include additional workflow activities to the queue. Each additional activity is defined as an `<additional-activity>` node, child of the `<queue>` node, its body containing

the fully qualified name of the workflow activity following the same syntax as the activity attribute of the <queue> node.

Example

```
<additional-activity>
outbound.Validation_1st_Level
</additional-activity>
```

This feature allows users to access workitems across multiple workflow activities in a single queue. The permission defined on the queue applies to all eligible workitems in all activities for the queue.

<view> Node

The <view> node lets you set global views and is defined similarly to the <view> child nodes of the <queue> node. For more information, see "Views" within <menu> Node, on page 43.

<audit-types> Node

The <audit-types> node controls a list of predefined and deployment- defined audit message types, each of which is defines within a child <audit- type> node.

A <audit-type> child node has the following attribute.

Attribute	Required/ Optional	Description
name	Required	Name of the audit type.

Scrittura has the following built-in audit types, which determine the selection options available in the history window.

- user
- signature
- workflowedit
- edit-detail
- auto

<audit-user-actions> Node

The <audit-user-actions> node provides access to a list of user actions available from within the audit/history page. Audit user actions are defined within <audit-action> child nodes of the <audit-user-actions> tag.

A <audit-action> child node has the following attribute.

Attribute	Required/ Optional	Description
action	Required	Label of the audit user action, such as "Comment, "Phone Call", and so on.

<column-set> Node

The `<column-set>` node provides the ability to define a column set to be used by the different queues (see "Queue Columns" in <menu> Node, on page 43).

Column sets are defined within `<column>` child nodes of the `<column-set>` tag. A `<column>` child node has the following attributes.

Attribute	Required/ Optional	Description
display	Optional	Name that displays in the column header
variable	Required	PI variable that corresponds to the column
visibility	Optional	Column visibility in the queue screen. Possible values: `always-visible, always-hidden, default-visible, default-hidden`

Example

```
<column-set name="MyColumnSet">
<column display="Trade ID"
variable="tradeID" visibility="default-visible" />
<column display="Struct. ID" variable="tradeLinkID" visibility="default-visible" />
</column-set>
```

The column set can then be included in the queue element instead of the column elements:

```
<queue name="First Signature" activity="Scrittura.Signature_A">
<column-set name="MyColumnSet" />
…
</queue>
```

<search-columns> Node

The `<search-column>` node allows the definition of search variable in the Scrittura application. Those search variables are added to Scrittura FA Tables for optimized access in order to be available as search criteria.

> **NOTE:** Before adding a variable as a search variable, its FA Table details must be set up in the corresponding Product Definition.

The `<search-columns>` node has the following attribute.

Attribute	Required/	Description

	Optional	
exporting-tool	Optional	Reporting tool used to export search results. Possible values are JasperReport (for Jasper reports) or StyleReport (for Style reports).

Search variables are defined within the `<column>` child nodes of the `<search-column>` tag. A `<column>` node has the following attributes.

Attribute	Required/ Optional	Description
variable	Required .	PI variable referenced as a search variable.
display	Optional	Variable label as it displays in the search criteria dropdown list, or as column header.
quicksearch	Optional	Set to true to make this variable available in the Quick Search criteria. Default: `false`
hidden	Optional	Set to true to allow this variable to be added to the FA table without displaying as a search criteria. To be used for internal search variables not to be exposed to end users. Default: `false`
isarray	Optional	isarray Optional Set to true if this variable is an array. The cells of the array are searchable and their values added to the `SCRITTURA_FA_IDX FA` table. Default: `false`

Optional role specifications can be defined for a column by adding child `<role>` attributes and child nodes. For more information about the `<role>` attributes and child nodes, see <menu> Node, on page 43 .

<indexVariable> Node

The `<indexVariable>` node lets you specify which variables should be indexed in FA tables. The variables declared in this node are used for indexing when generating DDL (Data Definition Language) for the FA tables.

Example

```
<indexVariable>
<internalName>VAR1</internalName>
<internalName>VAR2</internalName>
</indexVariable>
```

NOTE: Only variables that are often used in searches should be indexed. While assigning many indexes to variables in FA tables might improve speed of searches, alternatively it might result in a database slowdown when the FA tables and variable indexes are being updated.

<reports> Node

The `<reports>` node allows for the specification of reports, each report being defined under the `<report>` child node.

A `<report>` child node has the following attributes.

Attribute	Required/ Optional	Description
name	Required	Defines the name of the report.
id	Required .	Defines the report identifier.
class	Required	Defines the class that implements the report.

<annotations> Node

The `<annotations>` node lets you specify the annotations that will be added by default to every Product Instance entering the workflow. Annotations can also be added by default to individual product definitions.

Each annotation thread is defined as a <thread> child node of the `<annotations>` tag.

A `<thread>` child node has the following attribute.

Attribute	Required/ Optional	Description
name	Required	Name of the annotation thread.

Roles can optionally be defined for each annotation thread in order to control which permissions are granted to users when accessing annotation threads. This is defined by `<thread-role>` child nodes of the `<thread>` node.

A `<thread-role>` child note has the following attributes.

Attribute	Required/ Optional	Description
name	Required	Name of the role given access to the thread.
access	Required .	Permission for this role in this thread. `read` or `write`

Example

```
<annotations>
<thread name="Drafting comments">
<thread-role name="" access="write"/>
```

```
</thread>
<thread name="Legal comments">
<thread-role name="" access="read"/>
<thread-role name="Legal" access="write"/>
</thread>
<thread name="Administrative comments">
<thread-role name="admins" access="write"/>
</thread>
</annotations>
```

<report-docs> Node

The `<report-docs>` node is required for multi-trade documents, such as HTML Multi-Chasers.

The `<report-docs>` node is defined by <template> child nodes. A `<template>` child node has the following attributes.

Attribute	Required/ Optional	Description
name	Required	Document name.
file	Required .	Appropriate template file and its location relative to the HTML template base directory.

<startup-classes> Node

The `<startup-classes>` node allows the specification of additional classes to be loaded on startup and on reload of the `scrittura-config.xml` file. These can be used to pre-cache data, initiate other systems, initiate the job scheduler, and start additional threads.

The `<startup-classes>` node is defined by <class> child nodes that define the name of the class to be loaded.

Example

```
<startup-classes>
<class>com.ipicorp.scrittura.util.LoadTagThread</class>
</startup-classes>
```

The following example shows a startup Java class that runs a new thread.

```
package com.ipicorp.scrittura.test;
public class TestStartupClass
{
public TestStartupClass() throws Exception
```

```
{
System.out.println("Startup up TestClass");
Thread t = new Thread(new TestThread(), "Test Thread"); t.start():
}
}
class TestThread implements Runnable { public TestThread(){}
public void run() {
for(int i =0; i < 10; i++)
{
System.out.println("Startup Test Class: " + i);
try
{
Thread.sleep(1000);
}
catch(InterruptedException e)
{
System.out.println("Test Startup Interrupted"); return;
}
}
System.out.println("Test Complete");
}
}
```

When the scrittura-config.xml file is reloaded, each of the startup classes attempts a restart. Any additional custom startup classes must include logic to gracefully handle such cases.

<saved-searches> Node

The `<saved-searches>` node allows for pre-defined search categories where users can save or access searches depending on their role.

Users with read access can run a report in the category. Users with write access can save, change, and delete reports in that category.

The `<saved-searches>` node has the following attribute.

Attribute	Required/ Optional	Description

allow-user-saves	Required	Boolean flag that provides the ability for all users to save their own personal searches under specified categories

Search categories are defined by <search-category> child nodes of the `<saved-searches>` tag. A `<search-category>` child node has the following attribute.

Attribute	Required/ Optional	Description
name	Required	Name of the category as displayed in the web user interface.

For each category, role permissions are specified by optional `<search-role>` tags, child of `<search-category>` nodes. A `<search-role>` tag has the following attributes.

Attribute	Required/ Optional	Description
name	Required	Name of the role.
access	Required	Access to the search category for users belonging to this role. Possible values are read or write

Example

Users in the `signers_b` role can read and write reports in the `Category 1` category. All users can run reports in the `Category 2` category. No user can save personal reports.

```
<saved-searches allow-user-saves="false">
<search-category name="Category 1">
<search-role name="signers_b" access="write"/>
<search-role name="signers_a" access="read"/>
</search-category>
<search-category name="Category 2">
<search-role name="admins" access="write"/>
<search-role name="" access="read"/>
</search-category>
</saved-searches>
```

At a minimum, the bare specification must be configured in `scrittura- config.xml.`

```
<saved-searches allow-user-saves="false"/>
```

<archive> Node

The `<archive>` node specifies the configuration of the Scrittura archiving model.

For full details on the configuration of the archive module and its capabilities, see Archiving, on page 274.

<search-queue> and <quick-search-queue> Nodes

The `<search-queue>` and `<quick-search-queue>` nodes specify the configuration of the Queue Style Searches that let you display the search results using the same user interface as the Scrittura bulk screens. This provides the powerful capabilities offered by the bulk screens.

For full details on this configuration and Queue Style Search capabilities, see Search and Reporting, on page 220.

Chapter 3: Product Definitions

The topics in this section describe the contents of Product Definition files and explain how to create them.

This section contains the following topics:

Product Definitions Overview

Each product implemented in Scrittura is associated with a Product Definition. The role of the Product Definition is to contain all data necessary in Scrittura for the product: it can be its economic data (also called Data Dictionary) or a list of technical fields internally used by Scrittura (such as, for workflow routing purposes) with no business meaning.

The data is used to populate the Product Instance of the trade once it is created in Scrittura. Amongst other information, Product Definitions contain the following data:

- Product variables, along with their audit and data types
- Workflow start activity
- Confirmation signatures
- Templates (HTML, JSP and WordML document generation only)

Although each Product Definition corresponds to a specific product, it is possible to organize the whole set of product definitions by creating sub- product definitions that contain common variables to be reused throughout the different product definitions. A specific Product Definition file, `commonvars.xml`, defines variables available for all product definitions and should contain all mandatory Scrittura variables.

All variables used by a product in Scrittura should be declared in its corresponding Product Definition (or a sub-Product Definition). This is where essential data like the audit type, the variable type (such as, currency amount, integer) are defined for the different variables. For example, the variable type defines the Java type of the corresponding Product Instance variable. Not defining a product variable in the Product Definition will not break the system, but this variable will default as a string variable and will not be audited.

Product Definitions are defined as XML files located under the `/products` repository of the Scrittura live folder. Once created, a Product Definition file must be declared in `scrittura-config.xml` under a `product-def` tag in order to be included by Scrittura into its configuration.

Product Definition XML File

The root node of a Product Definition file is ProductDefinition, which takes no attribute and has the following child nodes.

Child Node	Required/ Optional	Description
name	Required	The body of this tag specifies the Product definition display name.
shortName	Required	The body of this tag specifies the Product Definition internal name, as referenced by the application.
include	Optional	Include a sub-Product Definition into this Product Definition.
workflowStart	Required	Workflow process introduction step.
docmgr	Optional	DocManager hierarchy specific to this Product Definition, which supersedes the default one defined in `scrittura-config.xml`.
annotations	Optional	Annotations to be added by default to every instance of this product entering the workflow. Annotations can also be added globally to all products in `scrittura-config.xml`.
documentTypes	Required	This node defines the templates to be used for the different document types. It only applies to HTML, JSP and WordML templates.
signature	Optional	Definition of the different signature levels used by that product.
documentation	Optional	Textual description of the product, specified via its optional child nodes, text and url.
VariableDefinition	Optional	Definition for each variable of the product.
Views	Required	Section containing auto-generated views. These sections are read by the ViewGen command to generate JSP pages.

Sub-Product Definition Inclusion

Including a sub-Product Definition into a Product Definition is accomplished using the include tag. This tag references in its body the internal name of the sub-Product Definition to include (as specified by its shortName tag).

Example

The following XML Product Definition extract includes into the “FXO” Product Definition the sub-product definition whose short name is “BaseVars”.

```
<ProductDefinition>
<name>FXO</name>
<shortName>FXO</shortName>
<include>BaseVars</include>
...
</ProductDefinition>
```

Variable definitions in an included definition override the original Product Definition. For example, if you do multiple includes such as Product Definition A includes B includes C, and a variable is present in all three definitions, you get the variable as defined in C.

The included components are limited to variables, templates, and annotation definitions. View definitions are not included.

> **NOTE:** Sub-Product Definitions must also be declared under a `product-def` node in `scrittura-config.xml`.

Workflow Introduction Step

The workflow introduction step is specified by the `<workflowStart>` tag, which has the following attributes.

Attribute	Required/ Optional	Description
process	Required	Workflow process ID where trades will be injected.
activity	Required	Workflow activity ID where trades will be introduced.

DocManager Hierarchy

DocManager hierarchy allows for a specific Product Definition to override the default DocManager hierarchy and index mapping defined in `scrittura- config.xml` (respectively, by the `docmgr-path` root attribute and `dms- field-ref` nodes)

The entity-type hierarchy used to create the Scrittura Product Instance folder is in DocManager. Each element is the name of an entity type. If the first character is `"\"`, the folders are created in the library root. Otherwise, they are created as `"rootless"` folders.

The node to override the default DocManager settings is `<docmgr>`, which has the following attributes.

Attribute	Required/ Optional	Description
docmgr-path	Required	DocManager hierarchy of the Product Instance folder.

document-version-field	Required	Document version field.

The `<docmgr>` node accepts an unlimited number of `<dms-field-ref>` child nodes. These child nodes are required to specify the mapping of DocManager indexes against Product Instance variables or to override the default mapping defined in `scrittura-config.xml`.

`<dms-field-ref>` nodes follow the same schema as in `scrittura- config.xml` and have the following attributes. For full details about `scrittura-config.xml`, see Scrittura Configuration, on page 27.

Attribute	Required/ Optional	Description
name	Required	DocManager index name.
ref	Required	Product Instance variable mapped to this index.

Example

```
<docmgr docmgr-path="\Counterparty\product\Deal" document-version-field="3">
<dms-field-ref name="Counterparty" ref=cptyRef""/>
<dms-field-ref name="Producy" ref="productGroup"/>
<dms-field-ref name="Deal" ref="CommonReferenceID"/>
</docmgr>
```

Annotations

Annotations to be added specifically for this Product Definition are specified under the `<annotations>` tag, which does not have any attributes. Each annotation is defined by a single <thread> tag, and one or more `<thread- role>` tags that define access to the annotation thread.

A `<thread>` node has the following attribute.

Attribute	Required/ Optional	Description
name	Required	Name of the annotation thread automatically added.

A `<thread-role>` node has the following attributes.

Attribute	Required/ Optional	Description
name	Required	Name of the role given access to the thread.
access	Required	Permission for this role in this thread, whose possible values are read or write.

Example

```
<annotations>
```

```
<thread name="Drafting comments" />
<thread-role name="" access="write" />
<thread name="Legal comments" />
<thread-role name="" access="read" />
<thread-role name="Legal" access="write" />
<thread name="Administrative comments" />
<thread-role name="admins" access="write" />
</annotations>
```

Document Types

For trades using HTML, JSP or WordML document generation, valid document types and templates used for those document types are specified within the

`<documentTypes>` node, as `<DocumentType>` child nodes.

A `<DocumentType>` child node has the following attributes.

Attribute	Required/ Optional	Description
type	Required	Name of the document type.
baseTemplate	Required	The path of the base template. HTML templates should be stored in the templates folder of the live repository and must be manually loaded using the `SetConfig` command when changed. JSP templates should be stored in the custom folder and require redeployment when changed.

Signatures

Signatures can be specified for the product using the `<signature>` tag, one for each signature level. `<signature>` is an empty tag that has the following attributes.

Attribute	Required/ Optional	Description
level	Required	Signature level, specified as an alphanumeric character (A...Z or 1...n).
type	Required	Specifies the type of signatory whose possible values are `user` or `group`. The user name will be used as the signatory for signatures of type user, whereas the role name will be used for signatures of type `group`.

role	Required	Name of the role for this signature level.

Variable Definition

Variables are added to a Product Definition using the

`<VariableDefinition>` tag. A Product Definition file contains any number of `<VariableDefinition>` entities.

A `<VariableDefinition>` node has the following attributes.

Attribute	Required/ Optional	Description
valueType	Optional	Type of the variable. If not specified, the default value is `String`. Possible values: `Integer, Double, String, CurrencyAmount, Date, Boolean, LongString, OneOf, or Image`
audit-type	Optional	Audit class to log changes to this variable. Default value is `edit-detail` if not specified.
subscript1istLimit	Optional	Upper bound of the first array dimension for array variables. If no dimension is specified, the variable is a scalar.
subscript2ListLimit	Optional	Upper bound of the second array dimension for array variables.
no-default	Optional	Boolean value (set to `false` by default) that specifies whether the variable should be populated with a default value on Product Instance creation. It is mainly used for array variables. Once given a value, the element is not removed from the Product Instance if it subsequently becomes empty. The aim is to avoid unnecessary database inserts in the event of such large arrays or 2D arrays that may be mostly empty. **NOTE:** If set to true, there should also be no `defaultValue` child tag specified (see `VaraiableDefinition` child nodes in the following table), even an empty one. **NOTE:** The first array element is always inserted even when this attribute is set to `true.`
external-class	Optional	Fully qualified Java class name when variables should be stored in an external BLOB.

A `<VariableDefinition>` node has the following child nodes.

Child Node	Required/ Optional	Description
internalName	Required	Internal variable name used in conditions, workflows, and scripts.
visiblename	Required	Display name for the variable.
FATableName	Optional	Suffix of the FA table name where this variable should be stored, would it be set as a search variable. For example, specify DEFAULT to store it into `SCRITTURA_FA_DEFAULT`.
FAColumnName	Optional	Name of the FA table column where this variable should be stored, would it be set as a search variable. It is recommended to use the same name as specified by `internalName`.
editRole	Optional	Name of role granted edit rights on this variable. If not specified, all users have rights.
documentation	Optional	Documentation on this product variable (unused by Scrittura but available for XML readability purposes).
defaultValue	Optional	Default value for the variable, in case the variable is not populated in the incoming message. An empty tag is considered as an existing, but empty default value; hence the variable will be created in the PI.
VariableValidation	Optional	Validation criteria for the variable, as defined in the following table.
Widget	Optional	Specify the appearance of the control used when editing the variable using the `scrittura:edit` JSP tag, as detailed in the next paragraph

NOTE: `FATableName` and `FAColumnName` are only required when variables need to be added as search variables into the FA Tables.

A `<VariableValidation>` node allows the definition of validation criteria for the variable and has the following attributes.

Attribute	Required/ Optional	Description
onInitiation	Optional	`logAndWarn`, `refuse`, or `None (default)`. Specifies whether to warn on validation errors when the current value is null.
onEdit	Optional	`logAndWarn`, `refuse`, or `None (default)`. Specifies whether to warn on validation errors when the value changes.

A `<VariableValidation>` node has the following child nodes.

Child Node	Required/ Optional	Description
failureString	Optional	Display prefix for validation errors.
minValue	Optional	Minimum value.
maxValue	Optional	Maximum value.
minLength	Optional	Minimum string length (integer representing a number of characters).
maxLength	Optional	Maximum string length (integer representing a number of characters).

legalValues	Optional	Specifies the list of valid values. This node is generally optional, except for "one of" variables). This tag takes a single optional attribute, `className`, which specifies the class to use for the validation. Each valid value is under an empty child node, `LegalValue`, whose attributes are value (to specify the value) and label (to specify the label to display when using the scrittura:edit tag in the JSPs).
		`Example`
		`<legalValues>`
		`<LegalValue value="USD"`
		`label="United States Dollar"/>`
		`<LegalValue value="JPY"`
		`label="Japanese Yen"/>`
		`</legalValues>`

The `<VariableValidation>` tag specifies that if a data message enters Scrittura (`onInitiation`) with an invalid value for a specific variable, the event is logged and a warning is added to the screen for this data message (`logAndWarn`).

The three choices for `onInitiation` and `onEdit` are `none`, `logAndWarn`, and `refuse`.

If a user attempts to edit the value of a specific variable to pick an invalid legal value, the `onEdit=refuse` specification rejects this attempt. (Such an attempt would be highly unlikely, however, as this variable is specified to be displayed with a widget of type `dropDownList`.)

It is possible to specify a `className` attribute to the `legalValues` element in order to force validation against a separately generated set of valid values, such as a valid counterparty list from another database, made available through a Java class. This class must extend the `com.ipicorp.scrittura.remote.ValidValues` class.

`<Widget>` (child node of `<VariableDefinition>`) is used to specify the HTML control when editing the variable. <Widget> is an empty tag and has the following attributes.

Attribute	Required/	Description

	Optional	
type	Optional	Type of control to use, one of: `overrideDropDownList`, `dropDownList`, `multiChoiceList`, `checkbox`, `radioButton`, `textarea` or `text`.
rows	Optional	Number of rows for the control (default is 1).
cols	Optional	Number of columns for the control (default is 15).

Example Variable Definition

```
<VariableDefinition valueType="String">
<internalName>Trader</internalName>
<visibleName>Trader</visibleName>
<documentation>
<text>The trader associated with the trade.</text>
</documentation>
```

In this example, the XML sets the `valueType` String for the variable Trader that is to be displayed to end users as "Trader". The optional documentation section includes a short note for internal documentation purposes. This text could also refer to a URL, if an online documentation reference exists.

This variable will, by default, be displayed in views with a standard text box widget. This display specification can be overridden. No data validation takes place on this, as defined in the example.

Views

For data enhancement purposes, Product Definitions can include a Views section, specified as a collection of `DEViews` tags (Data Enhancement Views).

Although this feature is supported, it was designed to be used with trades whose documents are generated from HTML or JSP templates. Full details on this configuration can be found in previous versions of the documentation.

For WordML or DOCX trades, economic panels, custom panels, and data enhancement using document edition should be used instead.

Binary Large OBjects

Trades in Scrittura can have their variables (as defined in the Product Definition) stored in external large binary objects (BLOBs). However, the following access restrictions are imposed on these variables:

- Cannot be displayed in Queues.
- Cannot be displayed within the `<column display="" variable="" />` tags.

- Cannot be used in global or queue roles.
- Cannot be searched for within Scrittura.

Specify the external storage class in the Product Definition for all variables that should be stored in the external BLOB by setting the VariableDefinition external-class attribute to `com.ipicorp.scrittura.util.BlobExternalData`.

Example

```
<VariableDefinition
valueType="String" audit-type="NONE" external-class=
"com.ipicorp.scrittura.util.BlobExternalData">
<internalName>MyVariable</internalName>
<visibleName/>
</VariableDefinition>
```

Built-in Variables

The following table lists the built-in Scrittura variables defined for Product Instances (PI).

Additional built-in variables are defined for Structured Products, see Structured Products, on page 352.

Variable	Description
CommonReferenceID	Common reference ID, the internal unique Scrittura reference for a trade.
ProductDefID	Product type.
ProductDefDisplay	Display name of the product.
ProductInstanceVersion	Current version of the trade.
Signature[] SignatureDocx[]	Name of the person who signed the document. The edit control is a signature check box. This value can be set though scripting to force a specific signature. The Docx-suffixed variable is used when signing DOCX documents, the other one being for other document types.
SignatureName[] SignatureNameDocx[]	Name of the actual signer based on the role and signatory configuration. The Docx-suffixed variable is used when signing DOCX documents, the other one being for other document types.
SignatureTitle[] SignatureTitleDocx[]	Title of the actual signer based on the role and signatory configuration.

	The Docx-suffixed variable is used when signing DOCX documents, the other one being for other document types.
SignatureImage[] SignatureImageDocx[]	Signature image URL or reference corresponding to the actual signer based on the role and signatory configuration. The Docx-suffixed variable is used when signing DOCX documents, the other one being for other document types.
IsSigned[]	Boolean value usedto check whether the Signature[] array is empty.
workflowWorkitemID	Workflow workitem ID of the message.
BulkSignature	Similar to the Signature[] variable except that the behavior is tailored for bulk signatures.
USERID	Current user.
LastEditUser	Last user to edit the document.
LastForwardUser	Last user to forward the document to a new activity.
Last[TITLE OF DOC IN DOC MGR]Date	Time stamp when document is saved.
[TITLE OF DOC IN DOC MGR]Count	Count when a generated document is saved.
CurrentManualQueues	Comma-separated list of all the manual activities where a PI can currently be found. This variable is used primarily to return Search results. Using this in JSP code requires checking for characters within the string such as: `<scrittura:if condition='CurrentManualQueues.lastIndexOf (\"Executed\") != -1' />`
CurrentManualQueue	Name of the manual activity where the trade sits. This value is blank when a trade is in an automated activity.
CurrentManualProcess	Name of the workflow containing the manual queue where the trade sits.
scrHoldSequencedMessage	Boolean variable used by the sequencer and set to true when a Message Ticket must be held in the Sequence queue. For full details, see Message Sequencer, on page 290.

The following variables are available for Message Tickets (such as, when a message has just entered the workflow but has not yet been identified as a particular product type).

Variable	Description
cridExists	Boolean value to check whether the Common Reference ID already has a in the database.
workflowWorkItemID	Workflow workitem Product Instance ID of the message.

USERID, LastEditUser, and LastForwardUser

The USERID, LastEditUser, and LastForwardUser variables can be used to impose mutually exclusive functionality where the same user cannot edit a document and push it forward to an approval queue.

You can accomplish this by adding to the role definition for an activity in the workflow definition:

```
<role name="">
<condition>
<![CDATA[LastForwardUser <> 'USERID']]>
</condition>
</role>
```

or

```
<role name="">
<condition><![CDATA[LastEditUser <> 'USERID']]></condition>
</role>
```

The USERID variable is not available in JSP tags but the USERID variable is available in conditions. Therefore, you can use the following code in a JSP frame to determine if the current user is the same user who previously forwarded the document:

```
<%
// Ensure that the same user cannot save and forward the trade
// in both the previous and present queue
String userId = request.getUserPrincipal().getName(); ProductInstanceLocal pi =
(ProductInstanceLocal)request.getAttribute("pi"); String lastUser =
(String)pi.getValue("LastForwardUser"); boolean sameUser = false;
if(userId.equals(lastUser))
{
sameUser = true;
}
%>
```

[TITLE OF DOC]Count

When a generated document is saved, the following two product variables are updated:

```
Last[TITLE OF DOC]Date = now
```

```
[TITLE OF DOC]Count = [TITLE OF DOC]Count + 1
```

(or, 1 the first time around)

For example, for a document title of Chaser, you will get the variables `LastChaserDate` and `ChaserCount`. If `Title = Confirmation`, you will get the variables `LastConfirmationDate` and `ConfirmationCount`.

Spaces in the title are replaced with "_".

Document titles are defined in the product definitions. In the following Product Definition XML file excerpt, the title is Confirmation.

```
<documentTypes>
```

```
<DocumentType type="Confirmation"
```

```
baseTemplate="bt_CollarTest.html"/>
```

```
</documentTypes>
```

CurrentManualProcess and CurrentManualQueue

The `CurrentManualQueue` and `CurrentManualProcess` variables are often used with `<scrittura:if>` tags to check a condition.

The following is a code sample in a JSP file:

```
<table class="normal">
```

```
<scrittura:if condition='CurrentManualQueue.equals(\"Outstanding_Review_Confir
mation\") && CurrentManualProcess.equals(\"OtherDocuments\")'>
```

```
<tr>
```

```
<td class="cell" align="left">
```

```
<scrittura:label name="OutboundOCNextQueue" />
```

```
</td>
```

```
<td class="normal">
```

```
<scrittura:edit name="OutboundOCNextQueue" />
```

```
</td>
```

```
</tr>
```

```
</scrittura:if>
```

In this example, the `OutboundOCNextQueue` variable (both the label and value) are displayed in the table of the JSP page if the condition is true.

CommonReferenceID

The `CommonReferenceID` (also referred to as CRID) variable is the unique identifier for each PI. This variable is one of two mandatory variables that are required to create a PI in Scrittura. The other mandatory variable is `ProductDefID`.

This variable should be used as an internal reference to the trade. Business trade references should be stored in different variables.

ProductDefID and ProductDefDisplay

The `ProductDefID` and `ProductDefDisplay` variables are used in connection with Product Definitions. Like `CommonReferenceID`, the `ProductDefID` variable is a mandatory variable.

To map a tag file to a Product Definition XML file, the value for the `ProductDefID` variable in the tag file must map to the value in the <short name> tag in the Product Definition XML file.

Example Product Definition XML

```
<ProductDefinition>

<name>opbd</name>

<shortName>opbd</shortName>

...

</ProductDefinition>
```

In this example, the product opbd corresponds to the Product Definition `<name>` and `<shortName>`.

The `ProductDefDisplay` variable contains the long name of the Product Definition (defined in the `<name>` tag) the trade belongs to. It can be used to create folder indexing in DocManager upon creation of a PI.

Sample XML extract `scrittura-config.xml`:

```
<dms-field-ref name="Counterparty" ref="Counterparty"/>

<dms-field-ref name="Address" ref="cp_address1"/>

<dms-field-ref name="Product Type" ref="ProductDefDisplay"/>
```

In this example the configuration file maps the Scrittura variable `ProductDefDisplay` to a DocManager field "Product Type." This DocManager variable can then be used for indexing or as a folder title in DocManager.

Common Variables

Variables common to all products in a workflow may be defined in the `commonvars.xml` file, which must also be declared as a Product Definition in `scrittura-config.xml`. This sub-Product Definition is always included in all Product Definitions even if not explicitly added as an include tag.

Variable definitions in the `commonvars.xml` take precedence over similarly named variables in an individual Product Definition file.

At server startup, the Scrittura log warns of any situations where variable definitions seem duplicated. The same variable name may be defined across multiple Product Definitions, but these variables must also share the same datatype.

Detect Variable Changes

Scrittura stores one of the following change statuses for each variable.

Status	Description
0	Unchanged
1	Changed by the system (data feed, BSH, or classtool)
2	Changed by a user using the user interface

You can query change status by using the `isChanged` method of the `ProductInstanceLocal` interface, as shown in the following example.

```
int status = pi.isChanged("varName");
```

or

```
if( pi.isChanged("varName") > 0 )
{
changesToTrade = true;
}
```

You can use the change status information to route documents to different workflow activities based on changes to specific variables.

When a new version of a Product Instance is created, the change status is cleared for each variable.

This can also be done programmatically by calling the following:

```
pi.clearChangedFlags();
```

Array Handling

This section details how to specify and handle arrays in Product Definitions.

Specifying Arrays in Product Definitions

Scrittura can define variables as single or double indexes, that is, one or two- dimensional arrays. For example this might be required when listing an arbitrary number of holiday cities per security leg, or an arbitrary number of fax numbers for each counterparty to a trade.

Variables are specified as indexes or double indexes in the Product Definition using the `subscript1ListLimit` and `subscript2ListLimit` attributes.

Indexes can be alphabetical (A, B … Z) or numeric (1, 2, etc.). Each array index will begin at either the letter 'A' or the number '0' or '1'. It is however recommended to use integer indexes and populate arrays starting from index 1 rather than 0.

Once a variable is defined as a double array, each value in the array can be referenced in different ways depending on the language/context.

The following examples define a variable called `MultipleExercise`.

```
<VariableDefinition valueType="Boolean" subscript1ListLimit="3">

<internalName>MultipleExercise</internalName>

<visibleName>Multiple Exercise</visibleName>

<documentation>

<text>only for American/Bermuda swaptions</text>

</documentation>

<VariableDefinition>
```

The `subscript1ListLimit` value of 3 in this sample indicates that this definition supports values for an array of `MultipleExercise[1], MultipleExercise[2],` and `MultipleExercise[3]`.

To specify two-dimensional arrays:

```
<VariableDefinition valueType="String"

subscript1ListLimit="B" subscript2ListLimit="4">

<internalName>TradeLegHolCal</internalName>

<visibleName>Trade Leg Holiday Calendar</visibleName>

<documentation>

<text>array of city codes to apply to each leg</text>

</documentation>

</VariableDefinition>
```

This example sets up a two dimensional array of strings. The primary dimension is indexed as [A-B], the other as [1-4].

This corresponds to eight variables, usable in templates as `TradeLegHolCal[A][1]`, `TradeLegHolCal[A][2]`,... and so forth.

Specifying No Default Values

It is recommended that you do not set default values for each variable in an array; extra database overhead is required to initialize each variable in the array.

In order to do so, set the `no-default` attribute to true and do not include a `defaultValue` tag for this array variable. The array variable will then not be created upfront upon Product Instance creation.

The same principles apply to non-array variables, if you do not want to define them with a default value.

Referencing Array Variables

This section details how to reference arrays from within the Scrittura application (such as, JSPs, BeanShells, and so on).

Double-indexed Variables in the JSP Tag library

To use double-indexed variables in the tag library for JSP templates and views:

```
<scrittura:value name="Varname[IndexVar]" ...>
<scrittura:value name="Varname[IndexVar1][IndexVar2]" ...>
<scrittura:label name="Varname[IndexVar]" ...>
<scrittura:label name="Varname[IndexVar1][IndexVar2]" ...>
<scrittura:edit name="Varname[IndexVar]" ...>
<scrittura:edit name="Varname[IndexVar1][IndexVar2]" ...>
```

> **NOTE:** The double indirection is not supported. For example, the following will not work:
>
> `<scrittura:value name="Varnane[IndexVar[A]]">.`

Double-indexed Variables in HTML templates

In HTML-based scripts and templates, the Scrittura generation engine recognizes double-indexed variables using the following syntax.

`Varname[{IndexVar1}][{IndexVar2}]`

Double indirection is also supported, such as:

`Varname[{IndexVar[{Index2}]}].`

Double-indexed Variables in Beanshell

In BSH scripts, array indexing follows the following syntax.

- Integer indexes

  `Varname[IndexVar]`

  `Varname[IndexVar1][IndexVar2]`

- String indexes

  `Varname[eval(IndexVar)]`

  `Varname[eval(IndexVar1)] [eval(IndexVar2)]`

Double indirection is supported: `Varname[IndexVar[Index2]].`

Set up Categories of Variables with audit-type

In Scrittura, each variable can be categorized by "audit-type" for audit history reporting purposes.

The Scrittura application's **History** page lets you pick specific categories of variables of interest:

Scrittura distinguishes between build-in audit types and custom audit types (defined by customer and assigned individually to variables in product definitions).

To display audit types in the **History** page, add them to the scrittura- `config.xml` file (see attribute <audit-type> in `<audit-types> Node, on page 47`).

In the variable definition below, `AmortizationAmount[A]` and `[B]` are defined to be an economic audit-type:

```
<VariableDefinition valueType="Date"

audit-type="economic" subscript1ListLimit="B">

<internalName>ExerciseDate</internalName>

<visibleName>Exercise Date</visibleName>

</VariableDefinition>
```

Chapter 4: Workflow Configuration

The topics in this section describe how to configure and manage Scrittura workflows. They also describe the Workflow Reporting Module.

This section contains the following topics:

Workflow Manager

The Scrittura Workflow Manager provides a J2EE compliant workflow engine based upon the WFMC standard, also known as WMFC Interface 1. Refer to http://www.wfmc.org/ for details.

The Scrittura Workflow Manager consists of the following components:

- The Workflow EJB Objects
- The Workflow Engine Process
- The Workflow Modeler, a Microsoft Visio template (XPDL Workflow Model.vst) and stencil (IPI XPDL.vss) for creating workflow models and their XPDL description files
- The Workflow EJB client and Activity APIs

Workflow Manager Terminology

The following terminology is used in Workflow Manager.

- **Workflow Package**. A collection of workflow processes. A workflow package includes all the processes to be run by the workflow engine, and must be internally complete. Each application that includes the Workflow Manager will need to load a workflow package for the engine to run.
- **Workflow Process**. A formalized view of a business process, represented as a coordinated (parallel and/or serial) set of process activities that are connected in order to achieve a common goal. A process can be complete and stand-alone or can be a sub-process that must interact

with other processes to form a complete workflow definition.

In the Workflow Manager, each process must contain at least one start (work introduction) step and at least one end (delete) step.

- **Work Item (Workitem)**. The representation of the work to be processed in the system. A work item includes data, in the form of a remote work item reference and a property bag, but does not include workflow-process- state information. A work item can be active in more than one activity (as the result of a split) and in more than one process (as the result of a sub- flow call).
- **Work List (Worklist)**. The set of work items currently in an activity waiting to be processed. Typically a work list is presented to the user in a manual activity. Automated activities process items in the order they are received.
- **Activity (Step)**. A description of a piece of work that forms one logical step within a process. An activity may be a manual activity, which is not automated by the workflow engine, or a workflow (automated) activity.

  An automated activity consists of an activity that is performed by invoking an application as specified by the workflow designer. In this Workflow Manager, an automated activity can either be invoked in the workflow engine process (a classtool) or in the process of the owning application via a callback (an application). An automated activity can also be a predefined workflow action, such as delete or sleep.

  A manual activity is typically a workflow manual queue and served outside of the workflow engine.
- **Transition (State Transition)**. A point during the execution of a process instance where one activity completes and the thread of control passes to another, which starts.

  A transition may be unconditional, such that completion of one activity always leads to the start of another, or conditional, where the sequence of operation depends upon one or more Transition Conditions.

  Where multiple transitions exist, a single unconditional transition is considered an "otherwise" transition. All conditional transitions are evaluated first, and then the otherwise transition is considered.

  For an XOR Split, the conditions can be considered "if ... else if ... else if...else" expressions where the otherwise condition is the final else.

  For an AND Split, the otherwise condition is only considered if none of the conditional transitions are taken.

  Transitions can be cascaded using route activities to provide sophisticated routing logic, such as nested ifs.

Workflow Transitions Between Activities

A workflow is modeled in a Microsoft Visio diagram, where activities are linked with transitions.

Transitions may be defined to contain conditions. These conditions determine routing directions.

Conditions operate on variables in the Product Instance (PI) or Message Ticket passing through the workflow. The PI is the object that passes through the main workflows and contains links to documents and variables defined in Product Definitions. The Message Ticket is the object that

passes through the Message Processing workflow (see Message Processing Workflow, on page 281).

> **NOTE:** Variables must be specifically defined in Product Definitions to be used in the Scrittura workflows.

Workflow activities can be defined to run specific applications or classes (such as document generation or bean shell scripts). You can set parameters for these class activities directly in Visio through extended attributes.

Workflow Modeler

Activities and transitions are modeled using Microsoft Visio and saved in a Visio VSD file. A macro included with the Scrittura Visio stencil converts these activities and transitions into an XML file. Each generated XML file defines a package. The XML package file should be saved in the workflow folder of your Scrittura application. The generated XML package conforms to the standard DTD set forth by the Workflow Management Coalition (WfMC).

To use the workflow modeler

1. Access Microsoft Visio and open the XPD Workflow Model template. A new document is created with the IPI_XPDL stencil loaded.
2. Use the editor to create your workflow.

   Make sure that each activity and transition has a unique name and that the custom properties are configured for each activity and transition.
3. Save your document and use the IPI Workflow menu to generate your XML file.

   The XML file is saved in the same folder as your VSD document, with the same file name but the ".xml" extension.

   > **CAUTION:** The transition from Visio VSD to XML file format is not reversible. In other words, if you configure Scrittura by directly changing the XML file, Scrittura will accept your changes, but Visio will not read the changed XML file and visually represent your changes.Therefore, manual editing of the XML package contents is not recommended.

Different pre-defined shapes are available in the workflow stencil. Shapes are included into the workflow by dragging and dropping them from the stencil to the page body. They will subsequently be linked together, as per the requirements, using conditional or otherwise transitions.

Properties and extended attributes can be associated with the different workflow shapes in order to configure the workflow. Those can be accessed using the contextual Visio Shape Data window by right-clicking the shape and selecting **Data > Shape Data.**

Workflow Packages

The XML package file must be listed as a package in the workflow.xml file located in the Scrittura configuration folder. The workflow engine may simultaneously deploy multiple packages. Once a PI has been created from an incoming message, this PI enters a specific workflow package declared in the specific Product Definition.

The `workflow.xml` file lists package names and package file names to be loaded into the Workflow Manager. Each package requires a unique package name in `workflow.xml`. The name of the XML package file is derived from the Title of the Visio file as defined in Visio under `File > Properties.`

Workflow Engine Specifications

The following workflow definition features are either not supported by the Workflow Engine or they require special handling:

Graph Structure

Workflow Manager uses the "Free Graph Structure", meaning that AND Splits do not have to have an explicit matching convergence AND Join.

The Workflow Engine does not require the workflow processes to be acyclic. It also does not prohibit or check for recursion and explicitly supports one level of synchronous recursion back into calling a synchronous process from a sub- process (A calls B calls A). Additional levels of recursion may cause a work item to stall, waiting return, and never continue.

Transition Validity Checks

Transition conditions are not checked for syntactical validity until they are evaluated. A workflow definition is a flexible computer program. You can create a workflow definition that is syntactically correct and that the workflow engine can load successfully, but that is semantically invalid. Such a process may never terminate, may have conditions that never evaluate, or may enter infinite wait states.

Loop Activities

Loop activities are not implemented in the Engine. They can be implemented using an XOR Split and XOR Merge.

Inline block

The initial XPDL draft specification supports an item called an inline block.

These activities are not enforced by the Engine or supported by the Visio tool; such blocks may be modeled as sub-processes.

Workflow Process

Each page in the Visio diagram corresponds to a Workflow Process. A sub-flow may be used both as its own Workflow Process and as a sub-flow in another process.

Each Workflow Process must include a start (work introduction) step and at least one end (delete) step. These rules are required semantically in order for your workflow process to work, but are not detected or enforced by the Engine itself.

The default behavior for any activity with multiple inbound or outbound transitions is XOR behavior for all transitions.

With the Visio template, you do not need to use the XOR Split and Join graphical widgets unless you want to add clarity. AND Joins and AND Splits are supported on any activity in the engine but require the use of an AND Join or AND Split step in the Visio tool.

BeanShell

Transition conditions are evaluated using BeanShell (which is similar to Java).

Variables are evaluated in the following order:

1. Internal BeanShell variables.
2. Variables that are part of the workflow system (`arrivalTime` and `arrivalTransition`).
3. Variables placed in the `Workitem` property bag.
4. Variables retrieved though callbacks to the `RemoteWorkItem`.
5. Primitive types and their classes are automatically converted into BeanShell primitives, so you should not need to call `intValue()` for an integer, and so on.
6. Modeling objects in the XPDL itself, such as Applications, DataFields, Participants, and FormalParameters, are ignored. They are parsed, but never used. Package and workflow headers are not used.

For details on the XPDL grammar, refer to the XPDL documentation and DTD available on the WFMC website.

Extended Attributes and Settings

General workflow settings, as well as extended attributes for each workflow activity, are required for the Workflow Engine to interpret the workflow. Both are set as "Custom Properties" in the Visio tool.

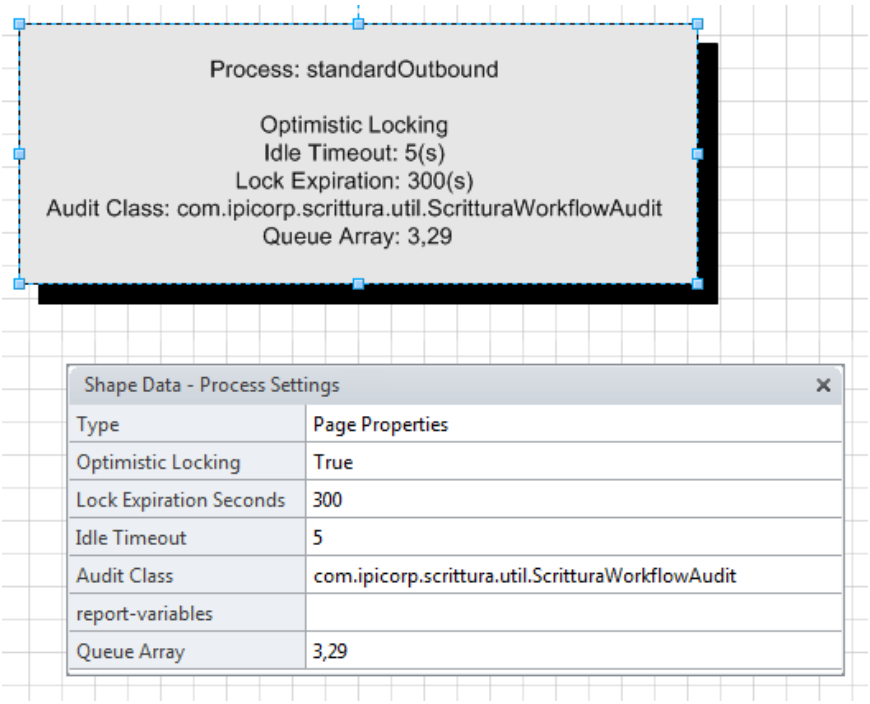
Process Settings

For each Scrittura workflow general settings must be specified. This is achieved using the Workflow Modeler within Microsoft Visio.

Open the workflow VSD file, which can be copied from the template provided with the distribution for new workflows. Make sure the stencil is loaded as well in order to access the pre-defined workflow shapes.

First set the name of the workflow by typing the process name in the Visio tab.

Other properties are set using the “Process Setting” shape available in the stencil: drag and drop it from the stencil to the page body and open up the associated Shape Data window. The following illustration is an example of “Process Setting” shape:



The following attributes are available as extended attribute and to be set using the Shape Data window.

Attribute	Description
Type	This has a fixed value of Page Properties.
Optimistic Locking	The Scrittura workflow engine and application support both "optimistic" (with a value of `true`) and "pessimistic" locking (with a value of `false`). Optimistic locking allows simultaneous access to a single work item, while pessimistic locking forbids simultaneous access. Should the workflow engine use optimistic locking, the application code must be designed to handle concurrent access and possible thrown exceptions. In the case of optimistic locking in the Scrittura application, a user is notified when a workitem is already opened by another user. The second user is still allowed access. The original user of the workitem is subject to having his or her work disregarded on save. In the case of pessimistic locking, the Scrittura application forbids a second user to open a given workitem unless the original lock has expired (as set using the `lock-expiration-seconds` attribute).
Lock Expiration Seconds	If optimistic locking is disabled, this configures the length of time in seconds a lock is held for. After the configured number of seconds elapses, the item is free to be locked by any other user.
Idle Timeout	Time, in seconds, that an activity thread should sleep before checking for waiting items. The lower this number, the more system resources the workflow engine will

	use. This number can be set low (5) for testing and set higher (15) in production when a slight delay between activities is acceptable.
Audit Class	The full name of a class that implements the WorkflowAuditor interface and should receive state change notifications from the workflow engine. Each process can have its own auditor class.
Report-variables	Coma-separated list of variable whose values will be logged in the tables `WF_RAW_STATISTICS` for reporting purposes, provided that the custom property `reportEnabled` is defined and set to true in `custom.properties`.
Queue Array	This field is automatically populated after setting up the workflow and cannot be manually changed.

Common Activity Settings

Most activities that can be inserted in the workflow accept the following properties.

Property	Description
Type	This property holds the type of the activity and cannot be changed.
Queue or Priority	"Priority" is a slight misnomer in that the Scrittura workflow engine does not treat workitems with a 1 priority as 'more important' than workitems with a 15 priority. Rather, the purpose of setting a priority for each activity is to pick a specific thread for the workflow engine to use when processing the activity. As a multi-threaded application, the workflow engine can perform many tasks at once. It is recommended that automatic activities that are less time-critical but more time consuming (such as document generation or Product Instance initialization) be segregated to their own thread. By separating these activities from manual activities where users expect fast GUI response (such as forwarding a document out of a manual queue), costly activities can be performed without impact on end-user perceived performance.
Report Enabled	Boolean attribute in order to include monitoring of this activity into workflow reporting.
Extra Report Variables	Coma-separated list of additional reporting variables for the activity that will be treated similarly to the list defined by the report-variables attribute in the Process Settings.
Description	An optional description of the activity.

Activities and Transitions

This section describes the different types of workflow activities available in the Scrittura workflow, how to configure them, and how to link them together.

Shapes are dragged and dropped from the stencil to the page body and their configuration is done using the Shape Data window.

> **NOTE:** In most cases, the properties detailed inCommon Activity Settings, on the previous page can also be defined but will not be repeated in the next section.

Start Activity

The Start activity is the explicit “Work Introduction” at the beginning of a workflow. This is the activity where new new work items created by an external client action are placed for workflow processing.

In the Workflow Manager, any activity that has no inbound transitions is considered a "Start" activity; therefore its use is optional, although recommended.

End Activity

The End activity is the “Delete” activity at the end of the workflow. All workflow process should contain a Start and an End activity. End is where the arriving instance of the work item is removed from workflow processing.

If this is the last instance of an item in the process, the process is considered complete and any processes waiting with the process are released. If this is the last instance of the item in the workflow package, the work item is removed from the workflow system.

Classtool Activity

A classtool activity allows the implementation of specific automated logic within the workflow. A classtool is based on Java class, either added to the implementation during the construction phase or already available as part of the core platform.

Since it is based on Java code, any change to the classtool logic would require rebuilding and redeploying the application.

The following properties can be configured using the Workflow Modeler.

Property	Description
Class	Full name of the Java class run by this activity.
Extended Attributes	Semi-colon delimited list of additional attributes that the classtool may require. This list is dependent on the Java class being used and is handled programmatically by that class. Extended attribute follows the sytax `{property}={value},` such as `title=Confirmation`.

BeanShell Activity

A BeanShell activity similar to a classtool allows the implementation of specific automated logic. It is however not based on a Java class but on a BeanShell script (BSH).

Unlike Java classtools, BeanShell scripts can be amended on-the-fly without restarting the application for the change to take effect.

From a performance perspective, Java classtools are far more superior than BeanShells. Therefore, Java classtools should be preferred to implement automated processing and can be used in conjunction with the BLogic Rule Engine to offer a high level of flexibility and configurability.

The following property can be configured using the Workflow Modeler.

Property	Description
Script	Name of the BeanShell script run by this activity

Subworkflow Activity

The subworkflow activity is used to call a sub-flow process within the Scrittura workflow.

A sub-flow process (or subworkflow) is a workflow process that is enacted or called from another (initiating) process (or sub process), and which forms part of the overall (initiating) process. Multiple levels of sub processes are supported.

A sub-flow process is useful for defining reusable components within other processes.

A sub-flow call may be either synchronous or asynchronous. When a synchronous sub-flow is called, the calling process waits for the sub-flow to complete before continuing. When an asynchronous sub-flow is called, the calling process continues immediately. Because the synchronization is a condition of the call, a sub-flow can be used both synchronously and asynchronously.

The following properties can be configured using the Workflow Modeler.

Property	Description
Process ID	Name of the sub-flow process to be called.
Activity ID	Activity to use as a Start activity in that sub-flow. It would generally be “Start”.
Synchronous	Select “SYNC” for synchronous sub-flows and “ASYNC” for asynchronous sub-flows.

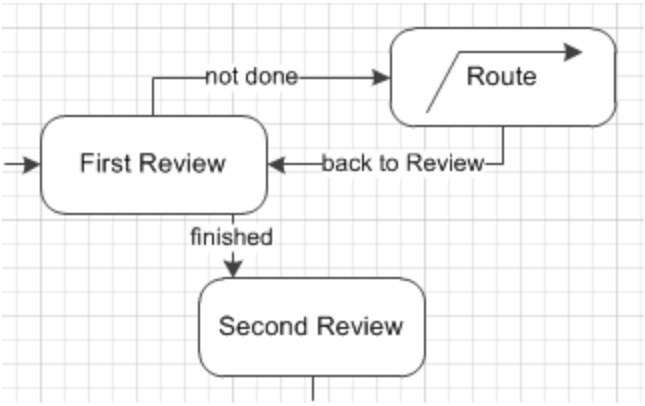
A subflow can be defined in a separate Visio file or in the same Visio file as the main process. To define the subflow in the same file as the main process, you must create a new page/tab with the name of the subflow. In either case, ensure that the subflow process name matches the process name defined in.

There are several possible reasons to define a subworkflow (or subprocess):

- Complex workflows with many activities/transitions
- Inter-department workflows (with each department responsible for maintaining its own workflow)
- Segregation for a clearer overview and easier maintenance
- Fewer dependencies when changes are required in the future

Route Activity

The route activity does not perform any processing and should be used when the next destination activity of an item is the same as the one it just left. The workflow engine does not indeed directly support looping back into the same activity and the route activity must be used for that purpose, as shown by the following illustration.



Sleep activity

The Sleep activity does not perform any processing, but holds the item for the number of seconds specified.

The following property can be configured using the Workflow Modeler.

Property	Description
Seconds	Time, in seconds, for the item to sleep

XOR Split and Join

The two XOR activities (Split and Join) are used to route trades within the workflow, based on exclusive outbound transitions. In the Workflow Manager, all activities except AND activities are implicitly treated as an XOR Split for outbound items or as an XOR Join for inbound items. Although recommended for workflow clarity, those activities are optional.

The XOR Split activity is a point within the workflow where a single thread of control makes a decision upon which branch to take when encountering multiple alternative workflow branches.

An XOR Split is conditional, and one (single) specific transition to the next activity is selected according to the outcome of the Transition Conditions.

The XOR Join activity is a point within the workflow where two or more alternative activities' workflow branches converge to a single common activity as the next step within the workflow. As no parallel activity execution has occurred at the join point, no synchronization is required.

AND Split and Join

The two AND activities (Split and Join) are used to perform parallel processing within the workflow. Any AND Split step should generally be followed later on by an AND Join step.

The AND Split activity is a point within the workflow where a single thread of control splits into two or more threads which are executed in parallel within the workflow, allowing multiple activities to be executed simultaneously. An AND Split can create additional threads of processing within the process.

An AND Split with mutually exclusive conditions cannot be logically differentiated from an XOR Split

The AND Join activity is a point within the workflow where two or more parallel executing activities converge into a single common thread of control. In the workflow engine, synchronization of an AND Join requires all threads within a process to converge before processing continues. If it is necessary to join some, but not all threads, a sub-flow must be used.

Application Activities

Applications are built-in Scrittura functions provided with the core distribution. Similar to classtools, applications are based on internal Java logic.

Applications have very specific scopes that apply to different objects (Message Ticket or Product Instance) or to parts of the workflow (Message Processing, Outbound, or Inbound workflows).

The following applications apply to Scrittura Message Tickets.

Application	Description	Parameters
RunLogic	Runs a Bean Shell logic script file=script file	`file=script file`
ParseMessage	Parses a Raw text/XML message to its HashMap representation none	none
CreateProduct	Creates a new Product Instance from the message	none
CreateNewProductVersion	Creates a new Product Instance version from the message none	none
DeleteMessage	Deletes the message ticket. The message is detached from the workflow workitem and removed. none	none

The following applications apply to Scrittura Product Instance (PI).

Application	Description	Parameters
GenerateDoc	Invokes the ADG module for HTML template based document generation.	`type=document type setMaster=true/false`

NewVersion	Creates a new version of the PI.	none
Remove	Removes the PI and workflow item, but not DocManager files/folders.	none
RemoveWithDocs	Removes the PI, workflow item, and DocManager files/folders.	none
RunBSHLogic	Runs a Bean Shell logic script.	`template=template file`

NOTE: Most Scrittura modules offer workflow integration directly through classtools rather than pre-wired applications. Refer to the specific modules for full details.

For inbound image processing, Scrittura offers a package of classes targeting the identification and indexing of inbound TIFF images. These images may or may not include encrypted barcode values.

The workflow around such processing is designed such that an external process recognizes an eligible TIFF image and constructs a PI message to hand off to the Workflow Engine.

The following applications then act on that Product Instance and are targeted to inbound workflows:

- `InboundSetDocmgrDoc`
- `BarcodeMapping, BarcodeMapping2`
- `Matching, Matching2`
- `PublishStub`
- `PublishConfirm2`
- `SetPropBag`

Workflow Transitions

Workflow transitions allow items to move from one activity or another. Two similar objects are available in the stencil, "Conditional Transition" and "Otherwise Transition", which are specialized cases of workflow transitions.

A **Transition**, as represented in the stencil by a thin arrow, can apply a logical condition which is evaluated when the workflow engine decides where to send the workflow item next. The following property can be configured using the Workflow Modeler.

Property	Description
Condition	Condition associated to the transition, following the BeanShell syntax. PI or Message Ticket variables can be used by simply quoting their name in the condition. For example, the condition `nextDest='docgen'` will be evaluated to true if the variable `nextDest` is equal to the string `docgen`. Variables used by workflow transitions must be defined in the trade Product Definition XML file. If this part of the workflow (hence the transition) is used by all product types, it is recommended to add it to a sub-Product Definition included into the Product Definitions of all products.

The **Otherwise** transition (a thicker arrow) is used to indicate that a workflow item will move from one activity to another unless another transition logic is first found to be true. Otherwise transitions take no attributes.

Example Using the Workflow Modeler

This section describes the use of Visio to define a simple Scrittura workflow.

This simple workflow consists of the following main activities and the transitions between each.

- Generate Document
- Review
- Data Enhancement (if the user chooses to edit document variables)
- Dispatch (Fax) if a logical condition is met (for example, if `DispatchMethod = "Fax"` and `CounterpartyFaxNum` is set)

Get Started with a Visio Workflow Definition

You can build a workflow definition from scratch or from an existing workflow definition template.

To create a new workflow from a template, open the XPDL Workflow `Model.vst` template that already includes the Scrittura workflow stencil and some basic workflow elements.

To build a workflow from scratch

1. Open Visio and create a new drawing based on a blank drawing.
2. In the Shapes pane, select **More Shapes > Open Stencil**.

   Navigate to and open the Visio stencil **IPI XPDL.vss**. This file is typically located in the /workflow directory under the Scrittura live folder.

   A blank document opens with a variety of workflow elements in the left panel.

   > **NOTE:** If you are creating a workflow from the defined template, a Start, End, and Process Settings element are already included in your workspace. If creating a workflow from a new file, these initial elements are not automatically included and must be created.

Create a Start Point and End Point

Every workflow must have a start point and an end point. If you are not using a template, these points need to be created.

Add Activities

In this example, Scrittura executes the HTML-based Document Generation. This is a specialized class and the Visio stencil includes a widget specifically configured for this function.

In this procedure, you will add the following activities to the workflow.

- **GenDoc**
- **Review**, defined as a **Manual** shape
- **Data Enhancement**, defined as a **Manual** shape
- **Fax Dispatch**, defined as a **Class** shape and copies the generated document into a spooling directory for the fax server

Activities and transitions are created in the workflow simply by dragging and dropping them from the stencil to the document.

To add these activities to your workflow

a. Drag the GenDoc shape from the stencil and drop it onto the drawing page.

b. Drag the Manual shape and drop it onto the drawing page.

   Rename the shape to Review by double-clicking the shape and typing in the new name.

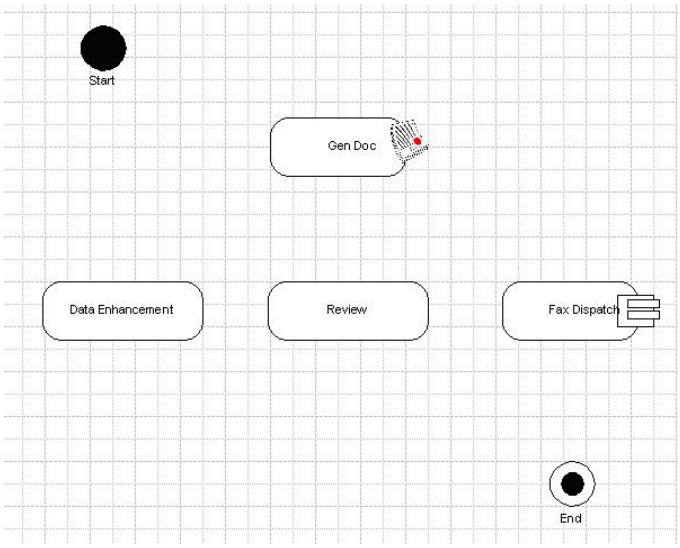
c. Drag the Manual shape and drop it onto the drawing page.

   Rename the shape to Data Enhancement by double-clicking the shape and typing in the new name.

d. Drag the Class shape and drop it onto the drawing page.

   Rename the shape to Fax Dispatch by double-clicking the shape and typing in the new name.

e. Move the shapes on your drawing page as desired, leaving room to add transitions. The relative positions of objects (activities, transitions, splits, joins, etc.) are ignored in generation of the Workflow XML, so you can position objects as convenient for you.

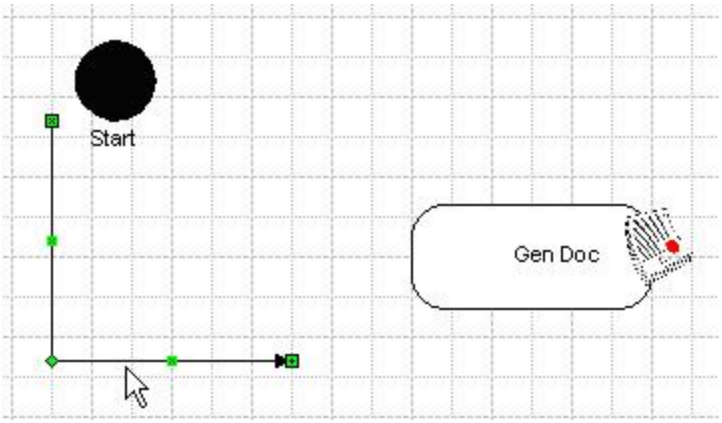


Add and Identify Transitions

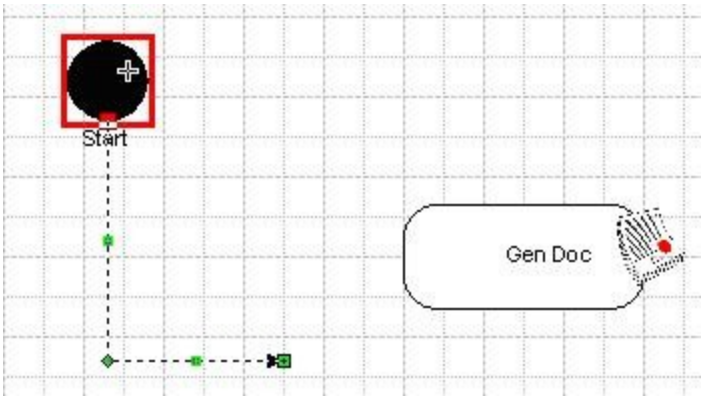
You must add transitions between each of these workflow activities and indicate the direction of the workflow transition.

To add and identify transitions

1. Drag the Transition connector shape from the stencil and drop it onto the drawing page.

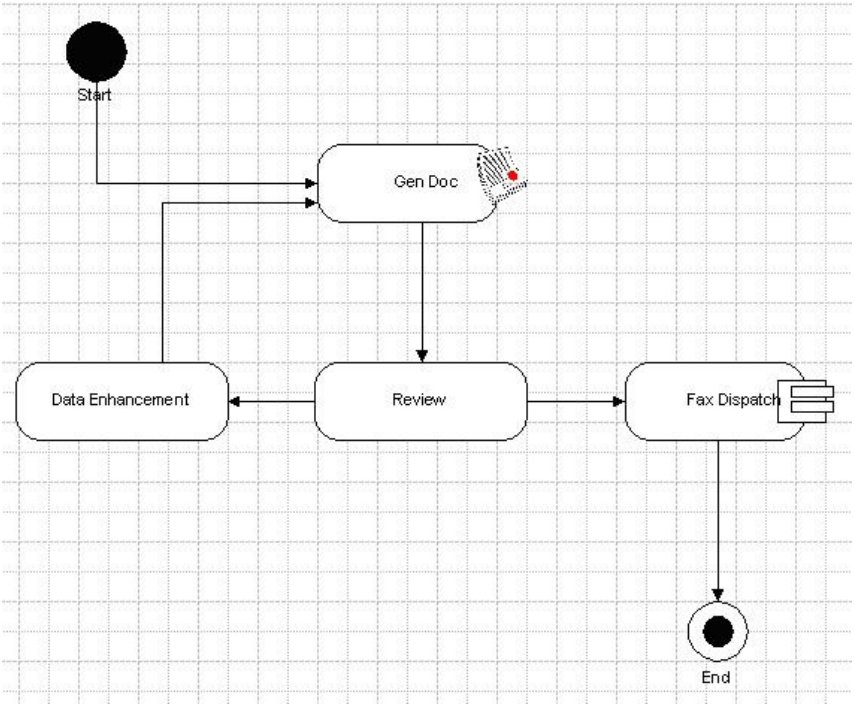


2. Drag the Transition connector so that the 'tail' is over the Start shape until the Start shape displays a red box, indicating that the connector is glued to the shape.

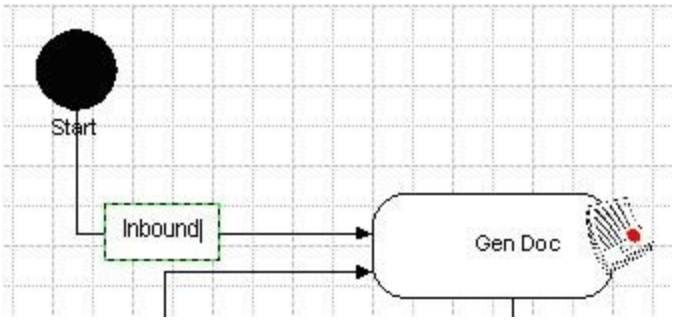


3. Drag the "head" of the transition arrow over the GenDoc shape until the GenDoc shape displays a red box, indicating that the connector is glued to the shape.
4. Create similar Transition connectors between the following:
   - GenDoc and Review
   - Review and Data Enhancement
   - Data Enhancement and GenDoc
   - Review and Fax Dispatch
   - Fax Dispatch and End

   The direction the arrow indicates the direction of the workflow transition.



5. Identify each Transition connector by double-clicking it and typing a name. The only requirement for these Transition names is that each be unique.

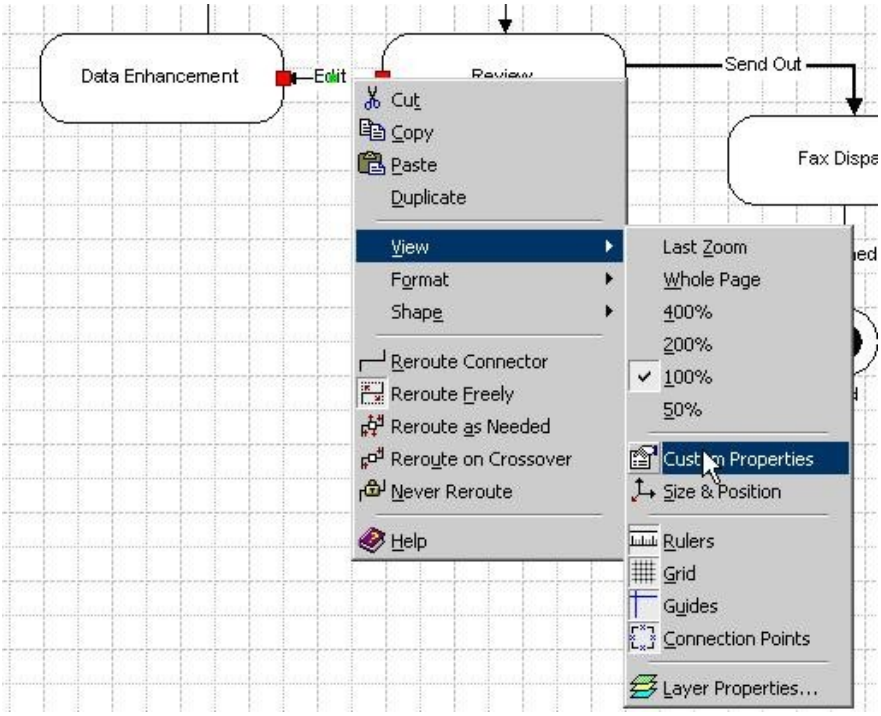


Define Conditions

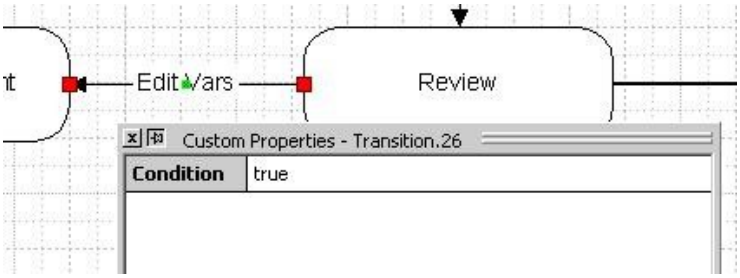
You must add the logic that determines the cases where workflow items will go from Review to Data Enhancement through the Edit Vars transition.

To add the logic

1. Right-click on the Edit Vars transition and select **View: Custom Properties**:

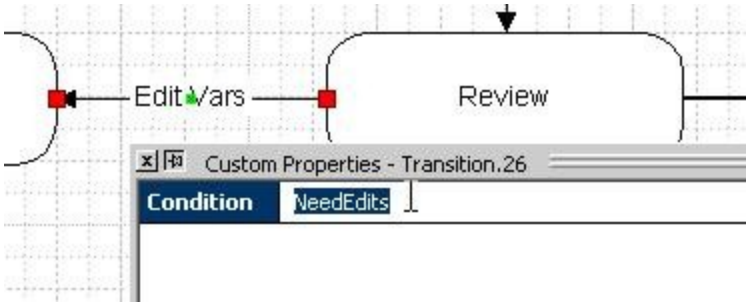


   The **Custom Properties** window displays with the option to set a Condition.



2. Replace the "true" condition with logic appropriate for the workflow. In the following illustration, the workflow will send documents to Data Enhancement where "Need Edits" is "true".

   "Need Edits" is a variable which would be included in a Product Definition (likely in the set of "CommonVars") and set to True or False when users review the trade. An assumption in the rule below is that "NeedEdits" is defined as a Boolean (true/false) variable.

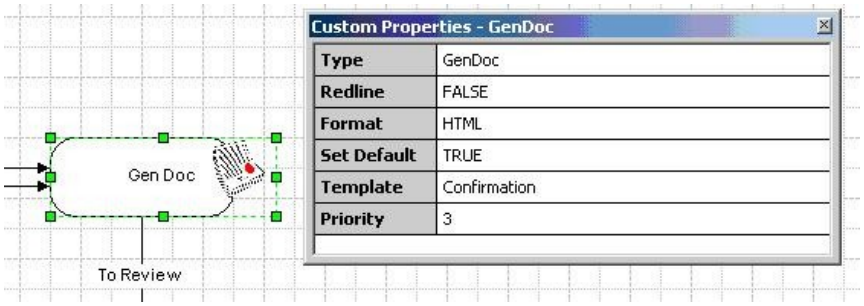


The syntax of the Conditions is flexible; other AND and OR examples are included in this document.

Define Classes and Extended Attributes

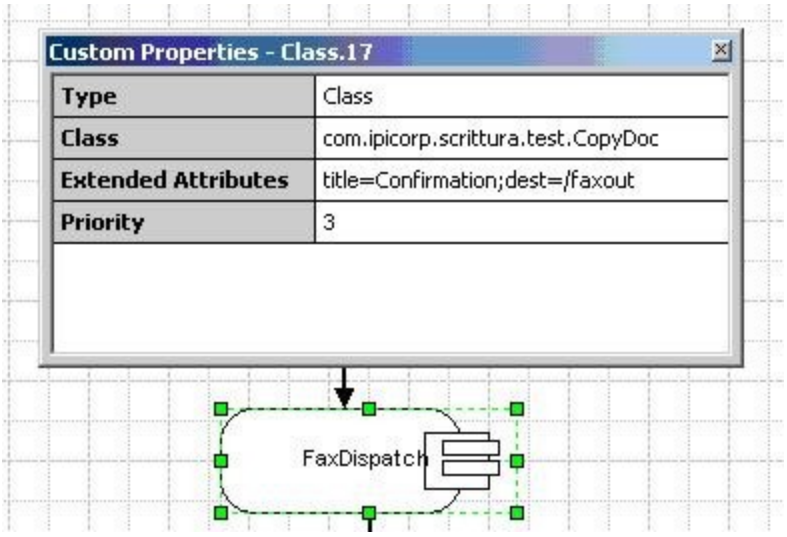
The GenDoc and Fax Dispatch workflow activities are defined by classes. The GenDoc class is built into the Visio Stencil.

1. From the Custom Properties window, click the GenDoc activity. The Custom Properties window changes to show you the class to be executed by the activity



The Fax Dispatch activity requires a measure of customization.

2. View its custom properties and specify the class **com.ipicorp.scrittura.test.CopyDoc** (which sends a copy of the generated document to a specified location).
3. Set Extended Attributes for this class: `title=Confirmation;dest=/faxout`. In this case, the CopyDoc class uses the Title to choose the document to copy and test for the copy.

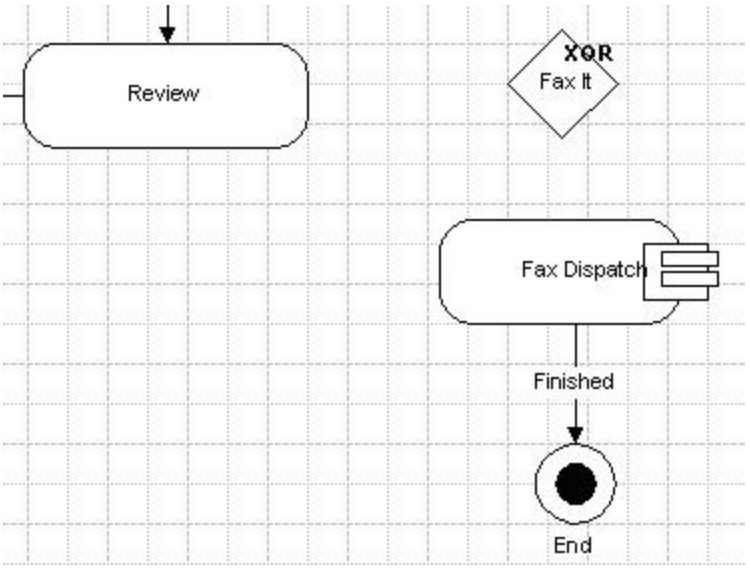


Other classes require other specific Extended Attributes; these requirements are documented elsewhere.

Define an XOR Split

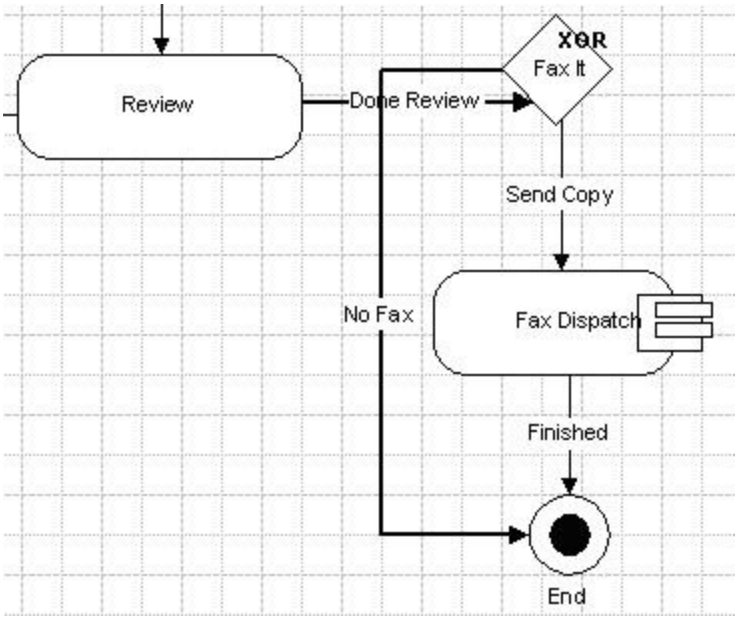
You must add logic only to send the fax if a fax number is set and if the `DispatchMethod` variable is set to fax. You must (again) replace the "Send Out" transition, this time by adding an "XOR Split".

1. Delete (again) the "Send Out" Otherwise Transition and drag in an "XOR Split" from the stencil. Give it the identifier "Fax It."

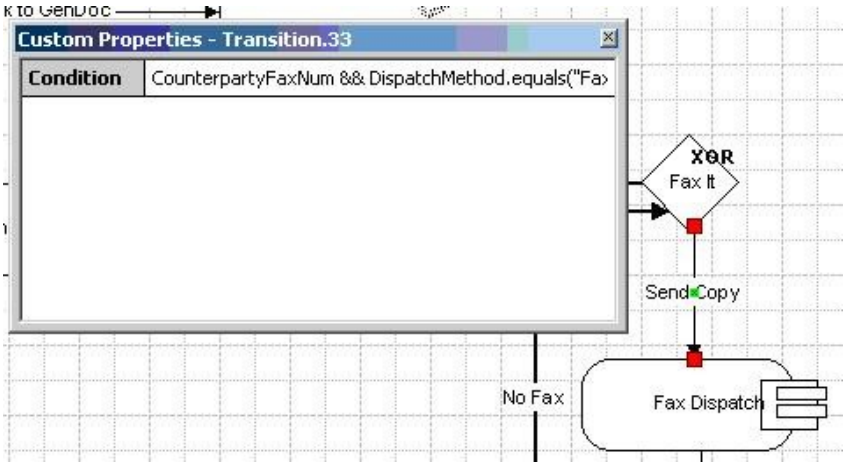


NOTE: XOR Split and XOR Merge functions may be synthesized without explicitly using the 'decision box' workflow elements as supplied in the stencil. By directly linking workflow steps, designers implicitly create these 'decision box' workflow features. These workflow elements are supplied for design convenience.

2. Create an Otherwise Transition from Review to the Fax It split to indicate that all workflow items go from Review to Fax It (unless NeedEdit was set to True).
3. Create a regular Transition from the Fax It split to the Fax Dispatch class. You will attach logic to this transition in another step.
4. Create an Otherwise Transition from the Fax It split to the End. This indicates the documents which are finished but not to be faxed are done in this workflow (but potentially still available in the Document Manager archive, depending on further configuration).
5. Give identifiers to each of these Transitions:



6. Add the condition to the Transition from Fax It to Fax Dispatch. Use double-ampersands to indicate "AND" (or double vertical lines "||" to indicate "OR").

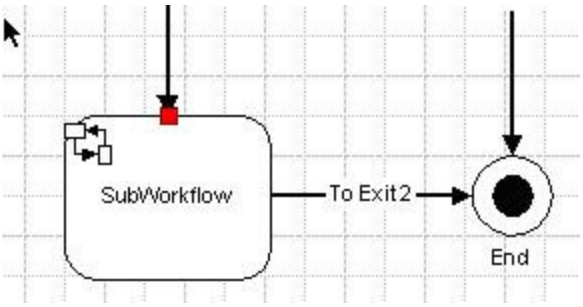


Link to a Sub-Workflow

To link a subprocess, make sure that the two processes are connected using the Visio workflow modeler.

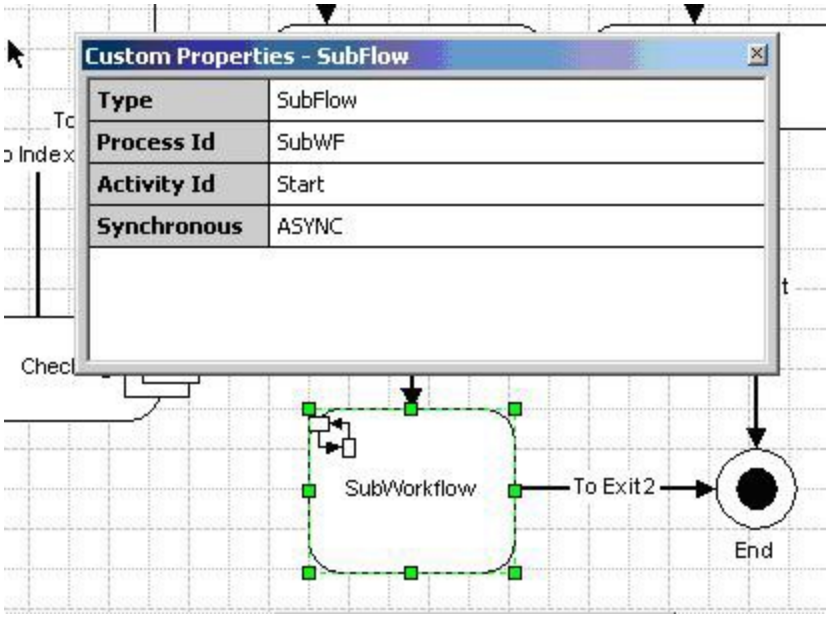
To link to a subprocess

1. Drag and Drop the 'Sub Workflow' Activity from the stencil and place it where the subflow should connect to the main workflow.
2. Assign a label name (for example, SubWorkflow).
3. Connect the inbound transition (that is, the transition from the last activity before jumping to the subflow) to the subflow.



4. Connect a transition out of the subflow to another activity or 'end' step.
5. Set properties for the subflow. Complete the following property fields:
   - **Process ID**. Define the name of subflow process (the label on the Visio page) .
   - **Activity ID**. Define the name of the target activity in the subflow (label of "Start" activity in subflow).

- **Subflow Type**. Asynchronous (ASYNC) or Synchronous (SYNC)

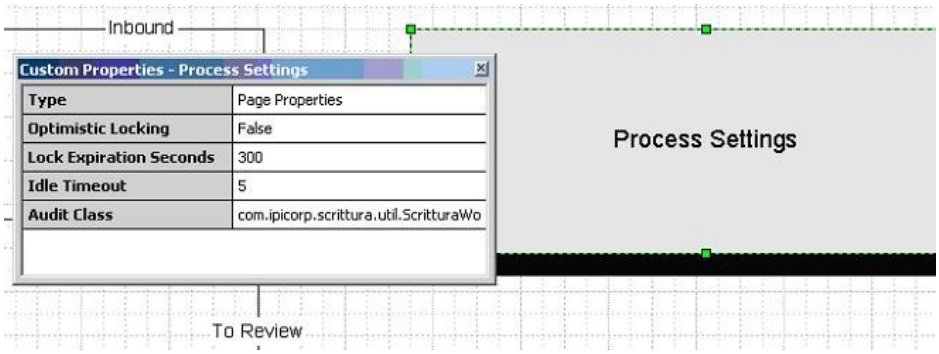


Define Process Settings

The final element to add to this simple workflow is the "Process Settings" element.

To add a Process Settings element

1. If this element was not already present in your template, drag this from the bottom of the stencil into your workflow area. This element sets the Extended Attributes noted in 0.
2. Select the large Settings area and View the Custom Properties.



3. Set the Audit Class to `com.ipicorp.scrittura.util.ScritturaWorkflowAudit;` this is standard code provided by Scrittura to create audit trails for all workflow activity. This selection could be altered or left null for different workflow processes if less auditing is required for the process.

   The Process Settings box may be resized and repositioned. After saving the Visio file, the box label changes to show the Custom Property settings.

Generate the XML

It is necessary to transform the Visio workflow into the XML format. The name of the output XML file is defined by the Title in the File properties of Visio.

To generate the XML

1. Open **File > Properties**, click the **Summary** tab, and define a title.

   Scrittura recommends that you map the xml file name to the name of the workflow process (tab on bottom left of page).

2. Generate the XML file (IPIWorkflow > Generate XPDL), which defines the workflow. If you defined your workflow by first opening a template or an existing definition file, you should already have an IPI Workflow menu available to generate the XPDL (eXtended Process Definition Language).

   If this menu item is not available, go to **Tools > Macros > IPI XPDL > XPDL Generator** and select **ExportXPDLFile**. This macro is loaded along with your original stencil.

The generated XML file will be saved alongside your VSD file, with the same base name. It should then be moved into your Scrittura environment and reloaded.

It is possible to add new transitions and activities to an existing workflow without disturbing existing documents and workflow items, but if activities are removed from a workflow definition, you risk orphaning existing workflow items.

Configure a New Workflow in Scrittura

It is possible to add new transitions and activities to an existing workflow without disturbing existing documents and workflow items, but if activities are removed from a workflow definition, you risk orphaning existing workflow items.

- `workflow.xml`
- `scrittura-config.xml`

> **NOTE:** You must also review all variables involved in the workflow to make sure they are declared in the corresponding Product Definition.

Once those changes are complete, reload the Scrittura configuration in the live system using the administration pages.

Amend workflow.xml for the New Workflow

Each workflow in a Scrittura configuration must be defined as a `<package>` in the `workflow.xml` file. This file lists the packages available; each incoming message can use a different workflow package based on its Product Definition.

The following XML extract is an example of workflow xml:

```
<workflow-packages
```

```
stop-on-error="false" items-per-pass="5" requeue-specs="120"
remote-factory="com.ipicorp.scrittura.util
.ScritturaRemoteWorkitemFactory">
<package name="Inbound.xml"/>
<package name="scr_test_msg.xml"/>
<package name="SubWF.xml"/>
<package name="swift.xml"/>
</workflow-packages>
```

The attribute `requeue-secs` sets the amount of time an item must be stalled before being requeued by the workflow engine. The default value for this attribute is 5 minutes (300 seconds). This parameter is optional.

You can configure `requeue-secs` to allow class tools to run for a long time (> 5 minutes) without having the items requeued by the workflow engine.

> **NOTE:** You must also configure the server JTA timeout (configured directly on the J2EE container) to permit this period of execution time before timeout.

Control Audit Behavior

By default, all audit types are audited unless otherwise specified in `workflow.xml`. Some types can be disabled by configuring the `<audit- config>` node of `workflow.xml`.

The following is an excerpt from a sample `workflow.xml` showing the workflow audit configuration.

```
<workflow-packages ...>
...
<audit-config>
<audit-type enable="false">workflow</audit-type>
<audit-type enable="true">economic</audit-type>
<audit-type enable="false">auto</audit-type>
<audit-type enable="false">edit</audit-type>
<audit-type enable="false">edit-detail</audit-type>
<audit-type enable="true">auto</audit-type>
<ref-type enable="true">RESID</ref-type>
<ref-type enable="false">CRID</ref-type>
</audit-config>
</workflow-packages>
```

The built-in audit types include the following:

- annotation
- auto
- data-init (linked to the checkbox "Show Initializations" on right side of the history screen)
- edit
- edit-detail
- user
- workflow

Other audit types may be specified in various product definitions. The built-in `ref-types` include the following:

- `CRID`: Product Instance audit messages
- `MSGTKT`: Message Ticket audit messages
- `RESID`: DocManager audit messages

Use the following to turn off all DocManager auditing:

```
<ref-type enable="false">RESID</ref-type>
```

Use the following to turn off all workflow auditing:

```
<audit-type enable="false">workflow</audit-type>
```

Use the following to disable all built-in auditing:

```
<ref-type enable="false">RESID</ref-type>

<ref-type enable="false">CRID</ref-type>

<ref-type enable="false">MSGTKT</ref-type>
```

Amend scrittura-config.xml for the New Workflow

Manual activities do not automatically appear as queues in the user interface where documents wait for end users. These queues (such as "Review") require additional configuration in the `scrittura-config.xml` file so that they will appear in the application itself. Make sure the manual activities are configured in this file.

The following XML extract is an example of manual queue definition.

```
<queue name="Review" activity="Outbound.Review">

<column display="Trade Type" variable="ProductDefID"/>

<column display="Trade Date" variable="TradeDate"/>

<column display="Counterparty" variable="Party[B]"/>

<column display="Amended" variable="TradeAmended"/>

<view name="Review"

type="custom" view="/jps/review.jsp" frameset="/jsp/viewframe.jsp"/>

<role name=""/>
```

```
</queue>
```

For details about how to configure queues in the application, see Configure Application Appearance, on page 159.

Workflow User Interface

Scrittura supports additional criteria to narrow the worklist displayed when opening an activity list, specified as part of the URL.

For example, a normal review activity hyperlink might appear as:

```
http://localhost:8001/scrittura/controller?e=queue&q=Scrittura. Data%20Enhancement
```

The following link adds the condition 'Party[B] LIKE 'B%'.

```
http://localhost:8001/scrittura/controller?e=queue&q=Scrittura.
Data%20Enhancement&sql=Party[B]%20LIKE%20'B%'
```

The syntax of the conditions is straightforward and Transact-SQL-like. Supported operators include the following:

- >
- <
- >=
- <=
- LIKE
- AND/OR (with support for parentheses)

This functionality can be leveraged in custom JSP pages that generate their own hyperlinks, setting the SQL element as needed.

Scrittura Workflow Architecture

This section summarizes how the different workflows are organized in Scrittura and provides details on the internal processing performed by the platform.

Workflow Organization in Scrittura

The global processing of a trade uses the following types of workflow:

- Message Processing workflow
- Main workflow

This section details the specifics of those different types of workflow.

Message Processing Workflow

The Message Processing workflow (available in the MessageProcessing.vsd workflow file) is a mandatory workflow used for trade preprocessing when a trade reaches Scrittura, such as an XML message.

Message Processing involves multiple steps, most commons ones being:

- Message parsing
- Data derivation
- Product or new version creation
- Message Ticket suppression

The purpose of this workflow is to capture the details of the incoming trade, perform some data derivation, and other steps in order to fully qualify the incoming trade. The actual trade, or a new version of it if the latter already exists in the system, is then created in the system.

When the trade message reaches Scrittura, a Message Ticket is created to handle the trade processing in the Message Processing workflow. Technically, a Message Ticket is a `MessageTicket` EJB that holds the raw data of the incoming message in a HashMap. Keys and values of the HashMap are of type string and extracted from the incoming message. The naming of the keys is free and does not have to follow the naming of the incoming message. Also, data being captured from the message is fully configurable and should follow the implementation needs and requirements.

Example

If the counterparty information is stored in the following part of the incoming XML message and the counterparty short code and long name must be captured:

```
<message>
<counterparty>
<shortCode>UNPTR</shortCode>
<longName>United Petroleum</longName>
<country>UK</country>
...
</counterparty>
...
<message>
```

The parser can be configured to capture that information, resulting of the following data stored in the Message Ticket HashMap:

Key	Value
`cptyShortCode`	UNPTR
`cptyLongName`	United Petroleum

When all the preprocessing of the Message Ticket data is complete, the actual trade (or a new version for existing trades) is created. This is achieved from within the Message Processing workflow by calling either the application `CreateProduct` (to create a new trade) or `CreateNewProductVersion` (to create a new version of the trade). During this step, the Product Instance (PI) is created (or amended), its repository is created, and the PI is injected into the workflow. That Product Instance will represent the trade in the main Scrittura workflows. For details on Product Instances and main workflow, see theMain Workflow, below.

Finally, the Message Ticket must be removed, which is achieved by calling the `DeleteMessage` application.

For more details on the steps involved in the Message Processing workflow as well as additional tools that can be used, see Message Processing Workflow, on page 281.

Main Workflow

A main workflow encompasses all workflows other than the Message Processing workflow, such as an outbound workflow, or inbound workflow.

In a main workflow, the trade is represented as a Product Instance (PI). Product Instance data is held in a HashMap, created from the original Message Ticket HashMap. Keys between the Message Ticket and the Product Instance are identical, the major difference being that values held in the Product Instance HashMap are converted into the type specified in the corresponding Product Definition.

> **NOTE:** It is recommended to declare all variables used by a PI in a Product Definition. Any variable not declared in a Product Definition will still be shipped in the PI but will have its type defaulted to string and will not be auditable.

Main workflows contain all the logic and queues necessary for trade processing. For more information about outbound and inbound workflows, see Outbound Workflows, on page 296 and Inbound Workflow, on page 309.

Any number of workflows can be created. Technical workflows can also be created, such as to process static data updates in an automated manner.

Workflow Engine Processing

This section summarizes the processing that takes place in the Workflow Engine and how it can interact with other Scrittura components.

- A JMS Message is posted to the `scrittura_tickets` JMS queue as a TAG file or XML file. This can be accomplished by dropping a file in the configured monitor folder. The trade drop monitor reads these files and posts them.
- A Message-Driven Bean (MDB) in Scrittura takes the message and creates a MessageTicket EJB (with the content of the message stored as a BLOB). This EJB is attached as a RemoteWorkitem to a new Workitem inserted into (by default) the "MessageProcessing" workflow at the "Start" activity. This specific activity and process can be overridden in the scrittura- config.xml file. Creating the new workitem posts a JMS message and triggers the workflow engine processing.

- The message processing workflow runs (MDB). Either some external action is performed (such as handling a fax response) and the MessageTicket is deleted, or a new Product Instance (PI) or new PI version is created.
- The workitem "moves" through the workflow by interpreting the logic defined in the XPDL files. Transitions between queues post new JMS messages, which are handled in turn by the MDBs. The Workflow property bag and/or the `RemoteWorkitem` handles requests for "Variables" by the workflow conditions. The remote workitems are "instantiated" by the `RemoteWorkitemFactory` class, specified in the `workflow.xml` file. Each "copy" of the workitem is an "ActivityItem" bean. This represents the state of the Workitem in a particular activity.
- When a version of a PI is created, Scrittura uses the WokflowItemID variable and finds all copies of the item in workflow currently. These are removed from the workflow. Then a new Workitem is created with the PI as the RemoteWorkitem for a new Workitem. This Workitem is put in the workflow at the activity and process defined in the Product Definition for the specific product type. A DocManager folder may also be created using parameters specified in the `scrittura-config.xml` file or in the specific Product Definition.
- Workflow activities can specify classes or applications to automatically run on the PI. State changes on completion cause JMS messages to be posted as new ActivityItems are created and old ones are removed.
- Manual processing takes place in queues attached to workflow activities. When a user opens an item, the `WorkflowClient.initiateProcessing()` method is called to "lock" the item. A Scrittura PI also has its own internal locking. When the user selects "Save & Forward," the workflow client API uses the `WorkflowClient.completeProcessing()` method, which changes the ActivityItem state and causes a new JMS message to be posted, waking up the workflow engine. Selecting "Close" runs `WorkflowClient.cancelProcessing()`.
- At some point, the workflow ends. Either the PI is explicitly deleted (optionally cleaning up DocManager) or the workitems just "drop out," leaving the PI around. In the latter case, the PIs are still available for search within Scrittura; in the former, they are not.

Workflow Reporting Module

The Workflow Reporting module is included in the standard distribution. A **cronjob** is available to run the report update on a daily basis.

Module Distribution Contents

The following table displays the contents of the workfow reporting distribution.

Folder or File	Description
`config`	Configuration files
`lib`	modules Libraries
`logs`	`scritturaReporting.log` log file

sql	sql scripts
templates	Reports templates
run.bat	Statistics calculation program
runReport.bat	Report generation program
ScritturaWFReporter.jar	module jar file

report-config.xml

The report-config.xml file is the main configuration file for the Workflow Reporting module. You can define database connection parameters and the statistics to be run, as well as configure reports to be generated using the statistics.

<WorkflowReportConfig> Node

The <WorkflowReportConfig> node is the root node of report- config.xml and has the following attributes.

Attribute	Description	Possible Values
deleteUnusedRawData	Specifies whether the system automatically deletes unused raw data.	true false
countBusinessDaysOnly	Specifies whether weekend and holidays are counted in duration.	true false
runReportGenAfterStatCalc	Specifies whether the report generation tool will run following the statistic calculation tool.	true false

<DatabaseConfig> Node

The <DatabaseConfig> node has the following attributes.

Attribute	Description	Possible Values
db	Defines the type of Scrittura database used.	mssql sybase oracle
driver	Defines the driver to be used to	weblogic.jdbc.sqlserver.SQLServerDriver weblogic.jdbc.sybase.SybaseDriver weblogic.jdbc.oracle.OracleDriver

	connect to the Scrittura database.	

url	Defines the URL fo rthe type of Scrittura database used.	`jdbc:bea:sqlserver://myDBServerName:1433`
		`jdbc:bea:oracle://myDBServerName:1433`
		`jdbc:bea:sybase://myDBServerName:2048`

user	Defines the username for the default Scrittura database user.	
password	Defines the password for the default Scrittura database user.	

<WorkflowStats> Node

The `<WorkflowStats>` node specifies statistics to be run against the workflow. The statistics are defined within the `<WorkflowStat>` child nodes of `<WorkflowStats>`.

A `<WorkflowStat>` node has the following attributes.

Attribute	Description	Possible Values
typeId	Defines the ID of the statistic. This value must be unique	0-6 (positive, whole integers)
name	Defines the name of the statistic.	`true` `false`
isSummary	Specifies whether this is a summary statistic	`true` `false`
runSchedule	Specifies the	`daily`

	frequency with which the statistic is calculated	`weekly`
		`monthly`
		`quarterly`
		`yearly`

maxDaysKept	Defiles the number of days the statistic will be kept	positive, whole integers -1 (default, always kept)
class	Defines the name of the class used to calculate the statistic.	

The start activity for a workflow statistic is configured by a single `<StartingActivity>` child node under each `<WorkflowStat>` node. The `<StartingActivity>` node has the following attributes.

Attribute	Description	Possible Values
process	Defines the name of the workflow process the activity belongs to.	
name	Defines the name of the activity that will be the starting activity for the duration.	
occurrence	Specifies whether to use the first time the activity was entered, or the last time when an activity can be visited multiple times by a PI.	`first` `last`

dateVariableToUse	Defines the name of a date- type variable that will overwrite the timestamp at which the PI entered the starting activity. Provides the ability to calculate duration from the "Trade Date" instead of the "Start" activity entry time.	Valid date-type PI variable

The end activity for a workflow statistic is configured by a single `<EndingActivity>` child node under each <WorkflowStat> node. The `<EndingActivity>` node has the following attributes.

Attribute	Description	Possible Values
process	Defines the name of the workflow process the activity belongs to.	
name	Defines the name of the activity that will be the starting activity for the duration.	
occurrence	Specifies whether to use the first time the activity was entered, or the last time when an activity can be visited	`first` `last`

	multiple times by a PI.	

<WorkflowReports> Node

The <WorkflowReports> node specifies the reports to generate and has the following attribute.

Attribute	Description	Possible Values
`controlFilesDropBoxPath`	Defines the same path as that defined in the `docmgr-config.xml` file for the element `<import-dir>` whose attribute `use-xml- meta` is `true`.	Unix style path

Reports are defined as `<WorkflowReport>` nodes and are children of the `<WorkflowReports>` node. A `<WorkflowReport>` node had the following attributes.

Attribute	Description	Possible Values
name	Defines the name of the report. This value is the title of the report once generated.	
id	Defines the ID of the report. This value must be unique.	As a convention, initials of the report name can be used.

runSchedule	Specifies the frequency with which the statistic is calculated. This value also represents the length of the timeframe of the report.	`daily`
		`weekly`
		`monthly`
		`quarterly`
		`yearly`

maxAge	Defines the time, in days, that the report is kept.	
queryClass	Defines the name of the query class used to generate the report.	
templateClass	Defines the report template to use (Jasper: `.xml` file; Style: `.srt` file)	

templateTool	Specifies the name of the vendor for the template.	`StyleReport`
		`CrystalReport`
		`JFreeReport`

exportPath	Defines the path to the folder where generated reports will be stored. The reports will also be copied from this folder into DocManager.	Unix style path
timeFrame	Use -1 as the value if the report is to calculate everything from the beginning until the last day ("Go Live date to date"). Example: If timeFrame=1, and runSchedule=weekly, report running for the day = 3/15/2014 Tuesday, then the report is for time between 3/7/2014~3/11/2145. Use another positive integer N if the report is to calculate everything that's N days/weeks/months/quarters/years before the report running day according to the runSchedule. Example: If timeFrame=2 and runSchedule=monthly, running on 3/15/2014 will return a report with the time frame: 1/01/2014~1/31/2014. -1 or greater integers	-1 or greater integers

The workflow statistics used by a report are defined as `<WorkflowStatID>` nodes and is a child of the `<WorkflowReports>` node. The `<WorkflowStatID>` node had the following attributes.

Attribute	Description	Possible Values
id	Defines the ID of the statistic used in this report.	

Product Instance variables to be included in a report are defined as `<WorkflowVariable>` nodes and are children of the `<WorkflowReports>` node. A `<WorkflowVariable>` node had the following attributes.

Attribute	Description	Possible Values
name	Defines the name of the Product Instance variable to include in the report.	valid FA variable
display	Defines the name of the variable the will be displayed on the report.	

type	Defines the type of the variable.	string, date

bucket	For date type variable only and for reports calculating an average, it is possible to create bucket.	`week`
		`month`
		`quarter`
		`year`

Activities to include in a report are defined as `<Activity>` nodes and are children of the `<WorkflowReports>` node. An `<Activity>` node had the following attributes.

Attribute	Description	Possible Values
name	Defines the name of the activity as defined in the Microsoft Visio workflow diagram. That activity must have the attribute report- enabled set to true in the diagram.	
display	Defines the name of the activity that will be displayed on the report.	

<WorkflowGroupReports> Node

The `<WorkflowGroupReports>` node specifies group reports. Group reports are defined within the `<WorkflowGroupReport>` child nodes of `<WorkflowGroupReports>`.

A `<WorkflowGroupReport>` node has the following attributes.

Attribute	Description	Possible Values
name	Defines the name of the report. This value is the title of the report once generated.	
id	Defines the ID of the report. This value must be unique.	As a convention, initials of the report name can be used.

runSchedule	Specifies the frequency with which the statistic is calculated.	`daily`
		`weekly`
		`monthly`
		`quarterly`
		`yearly`

maxAge	Defines the time, in days, that the report is kept.Not valid at this time.	
templateClass	Defines the report template to use (Jasper: `.xml` file; Style: `.srt` file)	
exportPath	Defines the path to the folder where generated reports will be stored. The reports will also be copied from this folder into DocManager.	Unix style path

The reports to include in a group report are defined as `<WorkflowReportID>` nodes and are children of the `<WorkflowGroupReport>` node. A `<WorkflowReportID>` node had the following attributes.

Attribute	Description	Possible Values
id	Defines the ID of the report. This value must be unique.	As a convention, initials of the report name can be used.

Run the Report

`run.bat` executes the main program to calculate the statistics whereas `runReport.bat` executes the main program to generate reports. Log properties are defined in `log.cfg`.

Condition Parser

This section provides details on the condition parser used for both conditional workflow transitions and the `<scrittura:if>` JSP tag.

Custom Parser

Scrittura can use two parsers to check conditions embedded in the `<scrittura:if>` tag or in conditional workflow transitions. One is the original parser using standard BeanShell syntax. The second is a custom parser that was created to evaluate conditions much more quickly and provide a more intuitive handling of null and empty values.

This improved parser will handle nearly all BeanShell conditions. Unparseable conditions will fall back to BeanShell evaluation and log warnings that should be forwarded to Scrittura Client Services for future support.

The Scrittura custom parser also accepts a simple syntax. For example, the following is a legal condition:

```
(a = b and c = d) or (e > f and c != r)
```

Missing values are treated as "", so conditions do not need to check for NULL values.

Valid Operators

The following operators are supported.

- =
- == (synonym for =)
- >=
- <=
- >
- <
- !=
- ( )
- !
- &&
- ||
- and (synonym for &&)
- or (synonym for ||)
- ~= - a case-insensitive equals sign.

For example,

```
var ~= "somestring"
```

is functionally equivalent to

```
var.equalsIgnoreCase("somestring")
```

You can also code a "not approximately equal" condition, similar to the following:

```
!variable~='string'
```

Quote Delimiters

Strings may be delimited either by single or double quotes. "abc" is equivalent to 'abc'.

Examples

Samples of valid parser syntax:

```
<scrittura:if condition="ATE = 'Not Applicable'">
```

```
<scrittura:if condition='manualEdit'>
```

In a workflow transition, with added support for "no" and "NO" and "No" (and "nO"):

```
stp and middleMarket~="no"
```

Handling of NULLs

In the custom parser, no value in an evaluation is ever NULL. Any missing value is represented as "".

Handling of Undefined Variables

The custom parser treats undefined variables as empty strings and compares them as such.

Supported String Operations

The following string operations are supported.

- endsWith
- startsWith
- equalsIgnoreCase
- trim
- toUpperCase
- toLowerCase
- indexOf
- lastIndexOf

Use of Escaping Quote Marks

Single quotes inside the double-quoted condition are properly parsed.

Double-quotes inside the double-quoted condition require the \" escape sequence.

Double-quotes inside a single-quoted condition must also be escaped.

Comparison of Different Datatypes

The custom parser is also able to compare non-equivalent datatypes. For example, integers can be directly compared to strings.

The following datatype promotions are attempted during evaluations.

Datatype of left side of evaluation	Datatype of right side of evaluation	Evaluation action
int	int	none required
int	float	convert int to float
int	string	parse string to integer and compare, parse error is false
int	date	always false
int	bool	convert into to bool using (int <>0)
float	int	convert int to float
float	float	none required
float	string	parse string to double and compare, parse error is false
float	date	always false
float	bool	convert float to bool using (float <>0)
string	int	parse string to integer and compare, parse error is false
string	float	parse string as double and compare, parse error is false
string	string	none required
string	date	convert date to string and compare
string	bool	convert string to bool using equalsIgnoreCase("true")
bool	int	convert int to bool using (int <> 0)
bool	float	convert float to bool using (float <> 0) - almost always true
bool	string	convert string to bool using equalsIgnoreCase("true")

bool	date	always false
bool	bool	none required
date	int	always false
date	int	always false
date	float	always false
date	string	convert date to string and compare
date	date	none required
date	bool	always false

BeanShell Scripting Syntax

BeanShell represents a scripting language for directly interfacing with Java beans. BeanShell is a small, free, embeddable, Java source interpreter with object scripting language features, written in Java. BeanShell runs standard Java statements and expressions, in addition to scripting commands and syntax.

BeanShell Scripts for Message Ticket Variables

In most cases, BeanShell script will access variables and built-in methods of a given Product Instance (PI) within a Scrittura workflow. In some cases, however, BeanShell scripts will need to access a received message before Scrittura has created a proper instance of the product. In these cases, called "Message Processing" stages, BeanShell has to access variables in the "tags" (otherwise known as a Message Ticket bean).

BeanShell Access to the Hash Map Table

When using BeanShell scripting in the message processing workflow, you can access the variables in the tag message, but not directly. They variables are accessed using a hash map table, but are not validated. Variables in Message Tickets are treated as strings and must be handled as such in BeanShell.

The standard syntax for scripts referencing variables in Messages is slightly different from the syntax for referencing variables in a formal Product Instance.

The following sample represents 'getting' a 'tag message' variable in a Message Ticket bean:

```
if(tags.get("DocType").equals("Master Agreement"))
```

This is in contrast to same IF condition in BeanShell when referencing a variable in a PI bean:

```
if(DocType.equals("Master Agreement "))
```

Additional sample code for comparing and setting hash map (tag) variables:

```
if(tags.get("VAR1").equalsIgnoreCase("true"))
{
```

```
tags.put("IgnoreMessage","false");
}
else if (tags.get("VAR1").equalsIgnoreCase("false"))
{
tags.put("IgnoreMessage","true");
```

Changing the CommonRefID in BeanShell

The CommonRefID (CRID) can be changed (in Trade Simulator or by means of BeanShell scripting), but only when operating on a Message Ticket bean. The CRID cannot be changed once the message is changed into a formal Product Instance using the built in applications *CreateProduct* or *CreateNewProductVersion.*

Creating an Autogenerated CommonRefID

The sample BeanShell script below can be used to generate a unique CommonRefID (CRID) during message processing workflow.

```
CounterTool ct = new CounterTool(); if(tags.get("DocType").equals("Master
Agreement") &&
tags.get("GetUniqueID").equals("Yes"))
{
cnt = ct.getNextId("orders");
while( (cnt % 5) != 0 || cnt < 1000)
{
cnt = ct.getNextId("orders");
}
tags.put("CommonReferenceID",
(String) (tags.get("PrependID")) + cnt);
}
```

> **NOTE:** CounterTool is the Scrittura counter class located in com.ipicorp.tools.remote. See Scrittura Counters, on page 379

In this example, the baseline being used for the new number is an existing number stored in the column "Counter" for the record with the Name "order". In this case, we increment by 5 and the first ID has to be at least the number 1000. The resulting CRID is the concatenation of PrependID and auto- generated ID.

> **NOTE:** The string "orders" is a randomly selected string. If you want to use the counter tool for other counting purposes and are replacing the string "orders" with the string "test" for example, a new counter will be created with the name "test".

Comparing Date Variables in Message Tickets

Suppose a trading system passes Message Tickets including a SettlementDate variable in the format "dd MMM yyyy" (that is, "15 Nov 2014"). Also suppose that the Product Instance in question is configured to include the variable SettlementDate, defined as a "Date" in the product variable XML definition.

Imagine a workflow configured such that BeanShell-based MessageProcessing code needs to compare the two SettlementDate values to detect if there has been a change.

A Message Ticket contains a hashmap of 'string' data. In this case, a direct comparison between the SettlementDate in a message ticket against the SettlementDate in a Product Instance will always fail with a datatype mismatch error.

The following BeanShell would be used in MessageProcessing to detect if a change had been made to SettlementDate in a new version of a trade handed off by the trading system. This BeanShell compares the SettlementDate variable from the MessageTicket as a date rather than a string. Then it can be meaningfully compared to the SettlementDate date variable in a Product Instance.

```
SimpleDateFormat formatter

= new SimpleDateFormat ("dd MMM yyyy"); ParsePosition pos = new ParsePosition(0);

Date settlementDate

= formatter.parse(tags.get("SettlementDate"), pos);

if( !settlementDate.equals(pihash.get("SettlementDate").value) )

{

// the SettlementDate changed in the message ticket -

// the original trade must have been amended

}
```

BeanShell Scripts for Product Instance Variables

This section details the use of BeanShell scripts with Product Instances.

String Comparison vs. Void Comparison

When comparing strings, you should use the ".equals" syntax; when comparing to 'void', use '==' syntax.

For example, the following BeanShell code works to make sure a value is not void but does contain just an empty string:

```
if (BulkAnnotation!=void)

{

if (BulkAnnotation.equals(""))

{

/* BulkAnnotation is not void but is an empty string */
```

```
}
}
```

NULL Values in BeanShell

Variables referenced in conditional logic in BeanShell either must first be declared with a default value (so that it is not 'null') or that condition needs to include a 'null' check.

For example, if the variable "test" is not given a default value in a Product Definition or in commonvars.xml, the following comparison will fail in BeanShell.

```
if(test.equals("value")) { ... }
```

To incorporate a check for null values, code as the following:

```
if (test!= void)
{
if(test.equals("value")) ...
}
```

It can be safer to initialize all Product Instance variables with some `defaultValue` in the Product Definition XML file, such as:

```
<VariableDefinition valueType="String" audit-type="status">
<internalName>test</internalName>
<visibleName>test</visibleName>
<defaultValue>value</defaultValue>
</VariableDefinition>
```

This BeanShell behavior applies to scripts run against both Product Instance beans and message ticket beans.

```
if (test!= void)
{
if(test.equals("value")) ...
}
```

Link from One BeanShell Script to Another

One can reference a BeanShell script in another BeanShell script using the `runBSHLogic` method on the Product Instance.

For example:

```
pi.runBSHLogic("scriptname.bsh");
```

This technique can be used to create hierarchies of scripts for an activity to execute, rather than a single script per activity. For example, one master BeanShell script could call others in sequence:

```
pi.runBSHLogic("PartyBTranslation.bsh");

pi.runBSHLogic("TranslateCurrencyCodes.bsh");

pi.runBSHLogic("TwoLetterCodes.bsh");

pi.runBSHLogic("ExpirationTime.bsh");
```

When a top level BeanShell sets or modifies a PI variable value subsequently accessed by a 'child' BeanShell' (or vice versa), only assign or reference variables using `getValue` and `setValue` methods directly on the product instances themselves.

For example, the following scripts should return "This did work, test is now value2."

ParentBeanshell.bsh:

```
test="value1"; pi.runBSHLogic("ChildBeanshell.bsh"); if (test.equals("value1"))

{

log.debug("This did not work, test is still value1.");

}

if (test.equals("value2"))

{

log.debug("This did work, test is now value2.");

}

if (test.equals("value3"))

{

log.debug("This did not work, test is now value3.");

}
```

ChildBeanshell.bsh:

The following script should return the log output: "This did not work, test is still value1." The child script won't know that test is value1, and on return the reassignment of "test" to 'value3' will not be seen by the parent script.

```
if (test.equals("value1"))

{

test="value2";

}

else

{

test="value3";

}
```

Instead, this should be written using setVar and getVar methods directly on the PI.

ParentBeanshell.bsh:

```
pi.setVar("test","value1"); pi.runBSHLogic("ChildBeanshell.bsh"); if test.equals
("value1")

{

log.debug("This did not work, test is still value1.");

}

if test.equals("value2")

{

log.debug("This did work, test is now value2.");

}

if test.equals("value3")

{

log.debug("This did not work, test is now value3.");

}
```

ChildBeanshell.bsh:

```
if (pi.getValue("test")="value1");

{

pi.setValue("test","value2");

}

else

{

pi.setValue("test","value3");

}
```

Use 'Else' to Map a Set of Variables in BeanShell

To translate a simple set of `beforeVar` value to `afterVar` values, such as "A" to "a", "B" to "b", and "C" to "c", the BeanShell script should use if ... else statements in order to maximize performance.

```
if (beforeVar.equals("A"))

{

afterVar = "a";

}

else if(beforeVar.equals("B"))

{

afterVar = "b";
```

```
}
else if(beforeVar.equals("C"))
{
afterVar = "c";
}
```

This is in contrast to the more direct (but potentially slower) syntax:

```
if (beforeVar.equals("A"))
{
afterVar = "a";
}
if(beforeVar.equals("B"))
{
afterVar = "b";
}
if(beforeVar.equals("C"))
{
afterVar = "c";
}
```

By using `else` statements as shown in the first example, the workflow engine can finish execution of the script more quickly should it find a match for one of the earlier choices.

BeansShell Functions

The following is a sample declaration of a simple function called testFunc, which converts a string:

```
testFunc(value)
{
if (value.equals("123"))
{
return "a123";
}
return value;
}
```

Calling the function:

```
newVal = testFunc(someVal);
```

There is currently no way in Scrittura to call a BeanShell function declared in one script from another script.

Example: String Manipulation and Tests

The following BeanShell works to check the value of the last character in a string.

```
if(foo.endsWith("t"))
{
bar=true;
}
else
{
bar=false;
}
```

The string class supports the following:

- equals()
- equalsIgnoreCase()
- compareTo() -- returns a number indicating collation order
- compareToIgnoreCase()
- endsWith()
- indexOf() -- to find location of a substring within (-1 if not there)
- lastIndexOf()
- length()
- startsWith()
- substring()
- regionMatches() -- compares specific parts of two strings
- toLowerCase()
- toUpperCase()
- trim() -- removes blanks from both ends
- charAt() -- gets one character at a specific position

Example: Setting a Date Variable

The following BeanShell example sets the variable `FlowDate` to the current date. `FlowDate` must be previously defined as a date in a Product Definition.

```
FlowDate = new Date();
```

The following BeanShell sets the `NextCallDate` variable (defined as a date in a Product Definition) to a date ten business days in the future. In this case, "business days" excludes Saturdays and Sundays but does not take into account any holiday calendars. This example uses the class `CalculateDays` located in the `com.ipicorp.scrittura.util` package.

```
NextCallDate = CalculateDays.addBusDays(new Date(), 10);
```

Example: Testing a Date Variable

Product Instance variables of `Date` type are initialized to January 1, 1900 when the PI is created. Therefore, to see if a date variable has been set in a Product Instance, you must check whether the date is more recent than 1/1/1900.

The following example uses the value -2120000000000 (in milliseconds, which is actually Mon Oct 27 18:06:40 EST 1902, but close enough to 1/1/1900 for these purposes).

```
if (someDateVar.getTime() > -21200000000001)

// mind the lowercase letter "L" at the end of the condition.

{

// the variable was set in the product instance

}

else

{

// the variable is using the default 1/1/1900 and wasn't set

// in the incoming message ticket

}
```

Log Activity in BeanShell

The following BeanShell writes directly to the `scrittura.log` file as it is executed, using the log4j class Category from the `org.apache.log4j` package.

```
Category.log = Category.getInstance("bsh");

Category.getInstance("bsh");

log.debug ("Hello there");
```

Chapter 5: DocManager Configuration and Administration

DocManager is a complete enterprise document management solution. It is a high-volume repository for all types of written and electronic documentation.

DocManager can be used in its built-in configuration, using the database or filesystem for document storage, or can be integrated with an external Document Management System.

This section contains the following topics:

DocManager Overview

DocManager is Scrittura's built-in Document Management System. It handles all operations related to document management.

DocManager can store three types of resources: Folders, Documents, and Links. A Folder resource behaves like a folder on the filesystem, in that it exists to store other resources (Documents, Links, or Folders). A Document resource is a meta-document that encompasses document metadata and the different versions of the document. Each version corresponds to an actual file and is handled in

DocManager along with its metadata. A Link resource behaves much like a document resource but in place of versions it contains a URL. It is not permitted for two resources to share the same name, type, and location.

Each Folder resource has an associated Entity Type (see the 'Entity Model' section below) which defines the types of metadata required or available for that Folder resource and its direct children (such as, Documents and Links stored directly within it).

All "Top Types" (entity types with no parent entity type) may either be located under the Library Root, a folder with Resource Id 0, or may take the form of rootless folders, in which case they have no parent. Folders under the library root are displayed in the DocManager tree (link), provided the user has sufficient permissions to see them, and are directly navigable within the user interface. Rootless folders are not visible within the DocManager tree (link) and can only be viewed in the user interface by searching for them.

All DocManager resources contain the following metadata.

Metadata	Description
Id	A unique identifier for the resource (database table primary key).
ResourceType	An indication of whether the resource is a folder, document, or link.
EntityType	The entity type to which the resource adheres.
ParentId	The id of the resource's parent (-1 for root folders).
NumVersions	The number of versions associated with the resource (0 for folders, 1 for links, 1 or more for documents).
CreatedBy	The user who created the resource.
CreatedOn	The user who created the resource.
ModifiedOn	The date that the resource was last modified.
Visible	Indication of whether the resource is marked as visible.
LogicalPath	A path representing the Resource's position in the DocManager hierarchy, formed of the resource's id and those of its ancestors.
DMSResourceLocator	The identifier for the associated resource in the external DMS, when using an external DMS with versioning enabled.
Index0-9	Ten searchable custom metadata fields; Index0 is reserved for storing a document's title. For how to add more index columns, see Add Index Columns, on page 391.

DocManager versions contain the following additional metadata.

Metadata	Description
Version	The version id (a unique identifier per resource).

DocExtension	The extension of the document.
CreatedOn	The date that the version was created.
CreatedBy	The user who created the version.
ModifiedOn	The date that the version was last modified.
ModifiedBy	The user who last modified the version.
FileSize T	The size of the stored document (after compression, if compression is used).
DmsVersionLocator	When using an external DMS, this represents the identifier for the associated resource/version in the external DMS.

DocManager can either be used programmatically through its API or interactively through its user interface.

DocManager Storage Configurations

DocManager can store all documents (including configuration files) in the database, and this is the recommended configuration. However, storing documents in a database is not a requirement. DocManager can also store documents on a file system, or in an external Document Management System. In these cases, the document metadata is still stored in the DocManager database but the file content is stored externally. For more information, see Linking to an External DMS, on page 143.

There are several reasons to choose to store the documents on a database including lower cost, faster retrieval times, easier backups and easier disaster recovery.

Any kind of document can be stored in DocManager. There are no proprietary document types.

Globalization of the document store is easy to achieve with an installation on a centralized web server. The web-based client is available anywhere on a network, and can even be exposed through a firewall, given proper authentication via SSL.

In its database or filesystem implementation, DocManager integrates with the full-text search capabilities of SQL, Sybase, and Oracle and can easily be integrated with existing applications through the DocManager API.

The documents can be stored in compressed form.

DocManager User Interface

The DocManager user interface consists of a menu navigation bar, and two content panes: the DocManager Explorer pane on the left and a resource properties pane on the right.

Page Navigation

The navigation bar just above the resource properties pane controls page navigation for DocManager. The page navigation includes an option to select the page you want to jump to, the

number of items per page, and a search field.

The values in the dropdown list for number of items per page are set in the `docmgr-config.xml` file. If a folder resource contains more items that the value selected, use the page number dropdown to view additional pages.

Type a keyword into the Search field to conduct a quick search for resources in the DocManager database. For more information about DocManager searches, see Search, on page 142.

The default and possible values available in the Items Per Page dropdown are configured under the `docmgr-tree-properties` node in `docmgr- config.xml.`

DocManager Explorer Pane

The DocManager Explorer pane (DocManager Explorer) is a familiar explorer- type tree that mimics the hierarchical resource structure. A hierarchical path is displayed just above the pane and specifies the path of the selected resource. When you initially access DocManager, the Library is selected in the DocManager Explorer by default.

You can expand and collapse each resource by clicking on the resource name. Expansion and contraction of the resource is indicated by a plus (+) or minus (-) icon to the left of the resource folder. The icon is hidden if a resource does not have children.

When a resource is selected in the DocManager Explorer, the contents or properties are displayed in the resource properties pane. Hover over a resource in the DocManager Explorer to display a context-sensitive menu with links to the basic actions that can be taken on the selected resource (see Core Operations, on page 140).

All configurations related to the DocManager Explorer are located under the `docmgr-tree-properties` element of `docmgr-config.xml.`

Virtual Folders

The specified number of items per page also determines the boundaries for virtual folder creation in the DocManager Explorer pane. If a selected folder contains more subfolders than the number of items per page, then the tree view is replaced by "virtual" folders, each representing a page of results. This is to avoid the hierarchical view from becoming unmanageable when there are large numbers of subfolders. Clicking on each virtual folder will expand (or contract) that folder in the DocManager Explorer and display the appropriate page of results in the Resource Properties pane.

Each virtual folder name is displayed as the name of the folder followed by a suffix in square brackets. Naming conventions are configured in `docmgr- config.xml`.

Resource Properties Pane

The resource properties pane displays the contents or properties for the resource selected in the DocManager Explorer. The default view for a folder is its contents. The default view for a document or link is the list of metadata information. A list of links is located at the top of the resource properties pane. These links let you perform various actions on the selected resource, or view aspects of the selected resource.

In addition to the core operations, the following links may be available:

- **Show Hidden**. Makes hidden files visible.
- **Hide Hidden**. Removes hidden files from the display.
- **View versions**. Displays a list of a document's versions.
- **Edit text**. Enables in-browser editing; only available when this functionality is enabled and the selected resource's extension matches one of the specified extensions.
- **Email Document**. Lets the user email the selected document.

  Limits: Available only when the emailing functionality is enabled.
- **Fax Document**. Lets the user fax the selected document.

  Limits: Available only when the faxing functionality is enabled.

For more information regarding core operations, see Core Operations, on page 140.

Drag and Drop

You can drag files from the desktop directly into DocManager. This is done by dragging the desired document onto a folder icon in the properties pane of DocManager. This immediately triggers the process for adding a document.

> **NOTE:** This functionality is available in Internet Explorer 10, Mozilla Firefox, and Google Chrome. It is not available in Internet Explorer 9 and earlier.

Entity Model

In DocManager, the Entity Model is used to provide structure to the document repository. An Entity is used to define a specific level of the library hierarchy. It is given a unique name, and includes the definition of the Indexes and Fields of the resources of that type. The Entity Model can be as strict or as loose as the implementation requires.

The Entity Model is defined in the `entity-types.xml` configuration file.

Indexes and Fields

Indexes and Fields are used as a means of describing and defining the resources stored in DocManager (Folders, Documents, and Links).

There are a fixed number of searchable indexes allowed in DocManager (10 by default), as well as an unlimited number of non-searchable fields that can be used to store additional data that describe resources.

The first index (index 0) is always used to store the title of a resource, be it a Folder, Document, or Link.

Indexes and fields are configured in `entity-types.xml`.

Entity Type Configuration

The Entity Model is configured in the entity-types.xml file. This section describes all the settings contained within this file.

An entity type is given a name and includes its indexes, fields, any child entity types, and can include an explicit Entity ACL (see Security Model Configuration, on page 139). Each entity type is defined by an `<entity-type>` element.

Example entity type

```
<entity-type name="Counterparty"
title-index="1" doc-title-index="9"
inherit-parent="false">
<index idx="0"
label="Title" index-type="folder" required="true" type="String"/>
<index idx="1"
label="Counterparty" index-type="folder" required="true" type="String"/>
<index idx="8"
label="Doc Date" index-type="document" required="false" type="Date"/>
<index idx="9"
label="DocType" index-type="document" required="true" type="String"/>
<field idx="0"
label="Address" field-type="folder" required="false" type="String"/>
</entity-type>
```

<entity-type> element

The <entity-type> element includes the following attributes.

Attribute	Required/ Optional	Description	Possible Values
`name`	Required	Defines the unique name of this entity type.	String
`title-index`	Required	Defines the number of the index used as the folder title of all folders of this type.	Positive integer
`doc-title-index`	Optional	Defines the number of the index used as the document title for all documents of this type.	Positive integer

`inherit-parent`	Required	Specifies whether folder resources inherit the indexes and fields of their parents.	`True` `False (default)`
`doc-form`	Optional	Defines a custom JSP index form that overwrites the built-in document index form. For more information on custom JSP indexes, see Custom Indexing Forms, on page 147.	String, such as `sampleDocFormJsp.jsp`
`folder-form`	Optional	Defines a custom JSP index form that overwrites the built-in folder index form. For more information on custom JSP indexes, see Custom Indexing Forms, on page 147.	String, such as `sampleFolderFormJsp.jsp`
`file-storage-identifier`	Optional	If running DocManager using an External DMS, this attribute allows specification of how documents will be stored. For more information, see Linking to an External DMS, on page 143.	Integer
`storage-location`	Optional	If running DocManager using an External DMS, specifies whether resources of the entity type should be stored in the database or the external DMS. By default (if this attribute is omitted) resources are stored in the external DMS if visible and under the library root, and otherwise stored in the database.	`database` - all resources of this entity type are always stored in the database external - all resources of this entity type are always stored in the external DMS, when possible.
`validator-class`	Optional	Defines a custom class to be used for validating Resources of this entity-type during creation/editing. For more details, see Custom Validator Classes, on page 137	

<index> element

The <index> element includes the following attributes.

Attribute	Required/ Optional	Description	Possible Values
`idx`	Required	Defines the unique number of this index.	0-9

`label`	Required	Defines the string to be displayed in the user interface for this index.	string
`index-type`	Required	Specifies the type of index.	`folder` - indexes apply to folder and documents (and hyperlinks) `document` - indexes apply only to documents (and hyperlinks)
`required`	Required	Specifies whether a value is required for this index. If an index is required, a resource will not be saved unless a value is supplied.	`True` `False`
`type`	Required	Defines the name of the validation type for this index. For more information on validation, see Field Validation, on page 147.	string

<field> element

The <field> element includes the following attributes.

Attribute	Required/ Optional	Description	Possible Values
`idx`	Required	Defines the unique number of this field.	positive integers
`label`	Required	Defines the string to be displayed in the user interface for this index.	string
`field-type`	Required	Specifies the type of field.	`folder` - This field applies to folder and documents (and hyperlinks) `document` - This field only applies to documents (and hyperlinks)
`required`	Required	Specifies whether a value is required for this field. If a field is required, a resource will not be saved unless a valid value is supplied.	`True` `False`
`type`	Required	Defines the name of the validation type for this field. For more information on validation, see Field Validation, on page 147	string

DocManager Configuration Files

In addition to the entity model configuration, DocManager stores its configurations in the following configuration files.

- `docmgr-config.xml` defines the core DocManager configuration.
- `mime.types` defines the list of mime types handled by DocManager.
- `mime.icons` defines the list of corresponding icons for the different mime types.

docmgr-config.xml

The `docmgr-config.xml` file is the main configuration file for DocManager. A full example of `docmgr-config.xml` can be found in the appendix.

The root node of docmgr-config.xml file is `<docmgr-config>`, and has the following child nodes.

- `<field-type>`
- `<import-dir>`
- `<configured-roles>`
- `<docmgr-tree-properties>`
- `<fax>`
- `<email>`
- `<document-compression>`
- `<full-text-search>`

<docmgr-config> node

The `<docmgr-config>` node is the root node of the `docmgr-config.xml`file. It contains a number of attributes and child nodes. The `<docmgr-config>` node has the following attributes.

Attribute	Required/ Optional	Description	Possible Values
allow-version-updates	Optional	Specifies whether it is possible to update existing versions of a document. This has particular significance for the import daemon where it enables the population of versions out of order	`True` `False (default)`

database	Required	Defines the database type used to perform full-text searches.	`oracle`
			`sybase`
			`mssql`

date-format	Required	If the DateValidator is configured to handle the validation of the index value of Date type, defines the format used to validate the value first, before trying to also validate with the other formats of. The class can be used in the validation configuration of the index values. After passing the validation, the value will be reformatted to this specified format.	MM/dd/yy MM-dd-yy dd.MM.yy MMM dd, yy yyyyMMdd yyyy-MM-dd yy MM dd dd-MMM-yy dd/MMM/yy MMM/dd/yy dd MMM yy dd MMMM yy MMMM dd, yy MMM dd yy MMMM dd yy MM dd yy yyyy-MM-dd
disable-security	Required	Specifies whether to disable security.	`True` `False`
docmgr-type	Optional	Specifies whether document content is stored in the database or in a configured external Data Management System. For more information, see Linking to an External DMS, on page 143.	database (default; documents are stored in the database) external (documents are stored external to DocManager)
edit-text-extensions	Optional	Defines the extensions for which in-browser editing is available. For more information see,In-Browser Text Editing, on page 143.	A comma separated list of file extensions Default list: `txt, bsh, bat, ini`
max-file-upload- size	Optional	Defines the maximum file size, in megabytes, permitted for a file to be stored in DocManager.	Positive integer Default: 100MB
number-of-indices	Optional	Specifies the desired number of index fields when more than 10 searchable fields are required. See Add Index Columns, on page 391.	Positive integer Default: 10

security-factory	Required	Defines the security factory class to use. The default security model is `com.ipicorp.docmgr.security.PerResourceAclFactory`.	String representing the fully qualified class name
stub-barcode-format	Optional	Defines the barcode format for the barcode stub when generating fax cover sheets.	`code39` (default) `code128` - improved recognition rates over `code39`
superuser	Required	Defines the user name of the system account on the application server. Allows security checks to be bypassed for the user specified.	String
use-entity types	Required	Specifies whether to validate resources against the entity model during creation/edition.	`True` `False`

<field-type> node

The `<field-type>` node defines permitted index/field types and associates each with a custom validator class (see Custom Validator Classes, on page 137).

Each `<field-type>` node must include the following attributes.

Attribute	Required/ Optional	Description	Possible Values
type-name	Required	Defines the string used to identify the validation class. This becomes the valid values in the `entity- types.xml` type attribute for indexes and fields.	String, such as "Zipcode"
validation-class	Required	Defines the java class that defines this validation type. For example, `com.ipcorp.scrittura.blogic.validation.BLogicValidator`	String representing the fully qualified class name

<import-dir> node

The `<import-dir>` node defines the import directories that are used by the Import Daemon (see Import Daemon, on page 149).

The `<import-dir>` node has the following attributes.

Attribute	Required/ Optional	Description	Possible Values

dir	Required	Defines the directory to be monitored for files to be imported into DocManager.	String representing the fully qualified path of the directory being monitored
use-xml-metafiles	Required	Specifies whether the directory contains control files or disparate resources	`true` `false`
sleep-seconds	Required	Defines the time, in seconds, between each read of the folder	Positive integer
workflow-name	Optional	Defines the name of the Import Daemon workflow.	String `DocMgr` (default)
workflow-activity	Optional	Specifies the Import Daemon activity.	`Image_with_no_XML` (default, use when `use- xml-metafiles` is false) `Doc_with_XML`

<configured-roles> node

The <configured-roles> node defines the groups configured for DocManager. This node is the basis of the DocManager permissions system, as individual permissions are assigned by role. Each role must be added to a `<role>` child node.

<docmgr-tree-properties> node

The `<docmgr-tree-properties>` node is optional and is used to configure the display properties of the DocManager tree.

The `<docmgr-tree-properties>` node has the following child nodes.

Child Node	Required/ Optional	Description	Possible Values
default-items-per-page	Optional	Defines the number of resources to display on a web page, and how many subfolders it takes before a folder is split into virtual folders within the DocManager tree	positive integer 100 (default)
virtual-folder-naming-scheme	Optional	Defines the appearance of the virtual tree nodes through its child nodes.	

A `<virtual-folder-naming-scheme>` node has the following child nodes.

Child Node	Required/ Optional	Description	Possible Values
format	Required	Specifies how the virtual	`head-tail` (the suffix is formed of the

		suffix is formed.	first and last letters of the boundary resources within it). `head` (the suffix is formed of the first letters of the boundary resources within it). `index` (the suffix is formed based on the number of the boundary resources, counting through in alphabetical order) .
head-length	Optional	Defines the number of characters to use when format is `head-tail` or `head`.	Positive integer
tail-length	Optional	Defines the number of characters to use when format is `head-tail`.	Positive integer

<fax> node

The `<fax>` node is optional and used to configure the Faxing-From- DocManager functionality.

The `<fax>` node has the following child nodes.

Child Node	Required/ Optional	Description	Possible Values
fax-enabled	Required	Specifies whether DocManager faxing is enabled.	`true` `false`
fax-class	Required	Defines a custom faxing class to be used by DocManager for the FaxDocument action.	String representing the fully qualified class name

<email> node

The `<email>` node is optional and is used to configure the emailing capability of DocManager.

The `<email>` node has the following child nodes.

Child Node	Required/ Optional	Description	Possible Values
email-enabled	Required	Specifies whether DocManager Emailing is enabled.	`true` `false`
email-class	Required	Defines a custom email class to be used	String representing

		by DocManager for the EmailDocument action.	the fully qualified class name

<document-compression> node

The `<document-compression>` node is optional and is used to configure DocManager to compress documents as the content is saved to the database.

Document compression is performed using the standard `ZipOutputStream` Java class, which uses the ZLIB compression library.

The `<document-compression>` node has the following attribute.

Child Node	Required/ Optional	Description	Possible Values
enabled	Required	Specifies whether compression is enabled	`true` `false`
default-level	Required	Defines the default level of compression	Positive integer between 0 and 9, inclusive -1 (default; no compression)
compression-on- extension	Optional	Specifies the level of compression for resources with a specific file extension. Can have one or more compression- on-extension nodes with required child nodes, extensions, and compression-level.	

A `<compression-on-extension>` node has the following child nodes.

Attribute	Required/ Optional	Description	Possible Values
extensions	Required	Defines a list of extensions to which this specific compression is applied.	Comma delimited list of valid file extensions
compression-level	Required	Defines the level of compression for the extensions defined by `extensions`.	Positive integer between 0 and 9, inclusive

<full-text-search> node

The `<full-text-search>` node is optional and is used to configure the full text search functionality.

The `<full-text-search>` node has the following attribute.

Attribute	Required/	Description	Possible

	Optional		Values
enabled	Required	Specifies whether full text searching is enabled	`true` `false`

`<full-text-search>` has a unique mandatory child node, `<full-text- parsers>`. `<full-text-parsers>` defines the parsers for reading document text for different document types, each parser being defined by a `<full- text-parser>` child node of `<full-text-parsers>`. There can be one or more `<full-text-parser>` nodes to define the different document types and each must include the case-sensitive attribute and the `<document- extension>` and `<document-parser>` child nodes.

The `<full-text-parsers>` node has the following attribute.

Attribute	Required/ Optional	Description	Possible Values
case-sensitive	Required	Specifies whether full-text searches are case sensitive.	`true` `false`

The `<full-text-parser>` node has the following child nodes.

Child Node	Required/ Optional	Description	Possible Values
document-extension	Required	Defines the file extensions of documents to be associated with the defined `document-parser` class.	Comma separated list of file extensions
document-parser	Required	Defines the full name of a class used to parse the text content of the document types defined by document-extension, such as `com.ipicorp.docmgr.fulltext .TextParser`	String representing the fully qualified class name

mime.types

Mime types are used by DocManager to give the browser an idea about what type of document is being requested from the server. The user's system can then try to open the document in the appropriate viewer for that document, depending on how those file types are associated on the user's system.

The DocManager API uses the `mime.types` file to associate file extensions with mime types. Additional mime-type/extension lists can be added to this file if necessary.

Example of the mime.types file extract

```
application/octet-stream bin dms lha lzh exe class

application/jsp jsp

application/oda oda

application/pdf pdf
```

```
application/postscript ai eps ps

application/powerpoint ppt
```

mime.icons

The `mime.icons` configuration file allows customized image icons to be associated with documents based on the document's extensions. These icons are displayed in place of the default icon in the DocManager user interface.

The configuration file is a plain text file with each line in the format `[extension]=[path to image file]`.

Example

```
doc=/docmgr/jsp/images/doc.gif

xls=/docmgr/jsp/images/xls.gif
```

Custom Validator Classes

When files are added or updated in DocManager, there is some default validation which is run to ensure that all the metadata marked as required for the appropriate entity type is present and correct (see Entity Type Configuration, on page 127). If these criteria are not satisfied, an error is thrown and the operation fails. Validation on file data and metadata can be augmented in two separate ways, through the use of custom validation classes.

Entity Type Custom Validation

For each entity type it is possible to define a custom validator class. This configuration is performed in the `entity-types.xml` file as the validator- class attribute of `<entity-type>` nodes.

This class allows additional restrictions to be enforced when attempting to create or update a resource of the corresponding entity type.

The defined class must implement the `com.ipicorp.docmgr.validation.ValidatorInterface` interface. This interface contains two methods, one for resource-level validation, and one for version-level validation.

Scrittura contains a class for BLogic: `com.ipicorp.scrittura.blogic.validation.BLogicValidator.` If this class is applied to an entity type, an error is thrown when a user attempts to upload an invalid BLogic rules file. For more information, see BLogic Business Engine, on page 203.

Index and Field Type Custom Validation

In the entity model, each index and field must specify a type; these types must be defined using a `<field-type>` node in `docmgr-config.xml.` Each of these types must specify a validator class.

The validator class must implement the `com.ipicorp.docmgr.validation.FieldValidation interface`. These classes define the validation to be performed on the values of the indexes and

fields associated with the corresponding types. Scrittura includes some validation classes for simple types such as String, Integer, and Date.

Security

The security model in DocManager is a configurable component. A new security model can be added to the DocManager API and then specified in the `docmgr-config.xml` configuration file.

The default security model in DocManager allows the flexibility for all resources in the system to be secured individually (Per-Resource), to be secured at the Entity level, or a combination of the two.

Access Control Lists

The Access Control List (ACL) is the basic element of DocManager security. An ACL is a list of the rights that can be granted in DocManager, and the groups that have been granted each right.

For every action a user tries to perform in DocManager, the DocManager API checks the ACL of the resource being accessed. The user is allowed to perform the action if the user belongs to a group that has been granted the necessary rights as defined on the ACL.

The default security model in DocManager combines Per-Resource ACLs with Entity Level ACLs. Per-Resource ACLs are ACLs defined for individual items stored in DocManager. Entity Level ACLs are defined for specific entity types and apply to any item of that entity type.

DocManager Rights

Rights are enforced within the DocManager API, which checks to see if the current user has been granted the appropriate rights when trying to perform any functions involving resources.

The following rights can be granted in DocManager.

DocManager Right	Description
Read	The most basic right in DocManager. A user must be given Read access in order to access any part of the DocManager library. A user with read access is allowed to view folders and documents, and use links. Read access is automatically granted if any other access type is granted.
Write	Lets a user edit index and field information for a resource, and set the URL of a link.
WriteDoc	Lets a user add new versions of documents (edit documents), add or replace the full text of a document, and migrate a version of a document from the DocManager database to an external DMS.
Create	Lets a user create new resources (folders, documents, links) in DocManager.
Delete	Lets a user permanently remove resources from DocManager.
Security	If a user is granted the security right on a resource, that user will see a "rights"

	menu option which lets the user set a Per-Resource ACL on that resource. Therefore, only users granted the security right are able to set a Per-Resource ACL. Security access automatically grants all other access privileges.

Groups and Users

Groups in DocManager are defined in the following areas.

- In docmgr-config.xml (see docmgr-config.xml, on page 130)
- In the deployment descriptors (xml configuration files) for both the DocManager API, and the DocManager Web App (the user interface).

  For DocManager deployed on WebLogic, groups are defined in the following files.

  - For the DocManager API: ejb-jar.xml, weblogic-ejb-jar.xml
  - For the DocManager Web App: web.xml, weblogic.xml
- In the application server

The following groups are defined in DocManager by default.

- readonly
- editors
- publishers
- admins

Security Model Configuration

The security model in DocManager is a configurable component and is configured in the main DocManager xml configuration file, `docmgr- config.xml`.The relevant settings are `security-factory`, `disable- security`, and `configured-roles`. For more information about these settings, see docmgr-config.xml, on page 130.

If using one of the default security models in Scrittura, permissions are primarily set by defining ACL on entity types (see Workings of the Default Security Model, below).

Workings of the Default Security Model

The default security model uses a combination of Per-Resource ACLs and Entity level ACLs. Entity level ACLs are defined in the Entity Type configuration file, `entity-types.xml`.

A default Entity ACL must be defined at the root level. Additional ACLs may then be defined on any Entity Type. For a complete example of the configuration file, see Sample: entity-types.xml, on page 420.

```
<dms-entity-model name="TestModel">

<entity-acl>

<read-list>

<group>admins</group>
```

```
<group>publishers</group>

<group>editors</group>

<group>readonly</group>

</read-list>

<write-list>

<group>admins</group>

<group>publishers</group>

</write-list>

...

</entity-acl>

</dms-entity-model>
```

Any user granted the security right can add `per-Resource` ACLs. `Per-Resource` ACLs are stored in the database. They are set on a given resource via the DocManager user interface. The "Reset Default Rights" button lets the user reset the current resource's ACL to the original default Entity ACL that applies to that resource.

`Per-Resource` ACLs, in effect, trump Entity ACLs. When the DocManager API checks to see if a user has the rights to perform a given function on a resource, it first checks to see if there is a `Per-Resource` ACL set for that resource. If none is found, it checks for an Entity ACL. If neither are found, the DocManager API checks the parents of that resource (above it in the tree) until it finds an explicit ACL, using the first `Per-Resource` ACL or Entity ACL it finds.

Examples

As a reference, see Sample: entity-types.xml, on page 420.

- If a resource of entity type "Deal" has no explicit Per-Resource ACL set, then the explicit "Deal" Entity ACL applies.
- If a resource of entity type "Product" has no explicit Per-Resource ACL set, and its parent folder has no explicit Per-Resource ACL set, then the "Counterparty" Entity ACL applies, because there is no explicit Entity ACL on entity type "Product."

Core Operations

The following is a list of the core DocManager operations, including a brief description of each. These operations can be executed using the DocManager user interface or the DocManager API.

- **Create Document**. Documents of different types (such as, fax, inbound, DTCC) can be created and saved at any location within DocManager. Create a document by clicking the “New” link in the user interface—not available for entity types which do not specify a doc-title-index. This operation requires create permissions on the parent folder.

- **Create Folder**. Create a folder within DocManager. Create a folder by clicking the “New” link in the user interface—only available for folders whose entity types have child entity types. This operation requires create permissions on the parent folder.
- **Create Link.** Create a link within DocManager that points to another document. Create a link by clicking the “New” link in the user interface— not available for entity types which do not specify a doc-title-index. This operation requires create permissions on the parent folder.
- **Delete Resource.** Delete any resource, be it Folder, Document, or Link. Delete a resource by clicking the "Delete" link in the user interface. Note that in the case of folders, their contents should be deleted before attempting to delete them in the user interface. This operation requires delete permissions.
- **Get Document Metadata.** Retrieves the indexes, fields, and other metadata of a given document. Get document metadata by clicking the "Properties" link in the user interface or, in the case of a document or link, by clicking the resource icon. This operation requires read permissions.
- **Download Document.** Retrieves document content. Download a document by clicking the resource name in the user interface. This operation requires read permissions.
- **Create New Document Version.** An arbitrary document can be uploaded as a new version of an existing document. Create a new document version by clicking the the “New Version” link in the user interface. This operation requires write-doc permissions.
- **Delete Document Version.** Delete an existing version of document. When viewing a document's version in the user interface, delete a document version by clicking the delete symbol associated with the desired version. If an intermediate version is selected, then higher versions are decremented accordingly. This operation requires delete permissions.
- **Edit Metadata.** Edit a resource’s metadata. Inherited indexes/fields cannot be edited. Changing the title-index for a folder or the doc-title- index for a document or link renames the document. Edit a resource's metatdata by clicking the “Edit” link in the user interface. This operation requires write permissions.
- **Cut and Paste/Move.** Deletes a document from one location and recreates it in another. If moving across entity types, indexes/fields need to be specified accordingly. Cut and paste/move a document by clicking the “Cut” and “Paste” links in the user interface. This operation requires write permissions on the resource and create permissions on the destination folder.
- **Copy and Paste/Copy.** Copies a document from one location to another. If copying across entity types, indexes/fields need to be specified accordingly. Copy and paste/copy a document by clicking the “Copy” and “Paste” links in the user interface. This operation requires read permissions on the resource and create permissions on the destination folder.
- **Change Rights.** Overwrite the ACL of the selected resource (see Security, on page 138). Change the rights of a document by clicking the “Rights” link in the user interface. This operation requires security permissions.

Search

DocManager provides three types of search functionality: Quick Search, Advanced Search, and Full Text Search.

- **Quick search.** Perform quick searches using the simple search box in the navigation bar of the user interface. A quick search searches DocManager for any resource whose title matches the selected text.
- **Advanced Search.** There is a link to advanced search in the DocManager title bar (the link is named “Search”). An advanced search provides additional search criteria. All searches are conducted by entity type, and any of the index columns defined for the selected entity type can be used to restrict the search.
- **Full text search.** Full text search lets allows the user search the content of uploaded documents for a particular word or phrase.

All searches conducted from the user interface are limited to a maximum of 50 results, although there is no hard limit when using the API. Clicking on any of the results in the user interface displays the properties for that result in the right pane, and displays the appropriate hierarchical tree in the left pane. This is the only instance in which rootless folders may be displayed.

> **TIP:** The case-sensitivity of DocManager searches is the same as the database case-sensitivity.The case-sensitivity setting for Scrittura trade searches does not apply to DocManager searches.

Full-Text Search

DocManager integrates with the full-text search capabilities of SQL Server, Oracle, and Sybase. SQL Server includes full-text search functionality as an option in the main install, and configuration is fairly straightforward.

A "Full Text Search" menu option is available on the DocManager user interface.

SQL Server Configuration

When installing SQL Server, make sure that the full-text search option is included—it may not be included in the default install.

Use the SQL Server Enterprise Manager to configure a full-text index on the `FILETEXT` column in the `DOCMGR_RESOURCEFILETEXT` table. To bring up the Full-Text Indexing Wizard, click **Tools > Full-Text** Indexing, or right-click the `DOCMGR_RESOURCEFILETEXT` table and select **Full-Text Index Table > Define Full-Text Indexing on a table.** Follow the steps in the wizard. Set up incremental population of the index to run daily, every 15 Minutes. If these menu options are grayed out, full-text search was not installed properly.

Make sure the Microsoft Search service and the SQL Server Agent service are installed and configured to run automatically. The SQL Server Agent service must run to perform the scheduled population of the full-text index set up in the Indexing Wizard.

Oracle Configuration

Configuration of full-text search in Oracle is similar to the process for SQL Server.

- When installing Oracle, do a custom install and include InterMedia and Java support.
- Configure a full-text index on the `FILETEXT` column in the `DOCMGR- RESOURCEFILETEXT` table.
- Set up incremental population of the index.

Sybase Configuration

Sybase allows the indexing and searching of `TEXT` columns. The `FILETEXT` column in the `DOCMGR_ RESOURCEFILETEXT` table is a `TEXT` column, and is the column to index for full-text search capability.

In-Browser Text Editing

It is possible to edit simple text files directly in the browser. This functionality is made available according to extension. The extensions to which this applies can be configured in `docmgr-config.xml` (see docmgr-config.xml, on page 130). If the `allow-version-updates` parameter is set to true, then this updates the latest version. If not, it adds a new version.

Linking to an External DMS

By default, DocManager stores both resource content and metadata in Scrittura's database. It is also possible to configure DocManager to store the content in an external Document Management System or filesystem. In this case, the metadata is still stored in DocManager's database, but all operations regarding the document content will be forwarded on to the External DMS.

External DMS Configuration

To use an external DMS for document storage, first the `docmgr-type` attribute must be set to external in `docmgr-config.xml` (see `docmgr-config.xml, on page 130`)

A specific connector to this external DMS must also be developed and integrated to Scrittura.

This external connector must implement the following interfaces, all located in the `com.ipicorp.docmgr.external` package:

- `IClientDMSResourceConnector`, for resource handling in the external DMS
- `IClientDMSResourceFileConnector`, for version handling in the external DMS

The connector has its own configuration file, `docmgr-external- config.xml`, which is a standard Spring file that holds all properties related to the integration with external DMS. It must contain the relevant beans in order to instantiate the concrete classes implementing the interfaces.

It is also possible to perform performance monitoring on the external DMS connector using a Spring aspect.

By default, only visible documents which are descended from the Library Root are stored in the external DMS. This behavior can be overridden using the `storage-location` parameter in `entity-types.xml` (see Entity Type Configuration, on page 127). Folders and links are never stored in the external DMS in any form.

Filesystem DocManager

The filesystem configuration of DocManager, where documents are stored in the filesystem, is a specific case of linking DocManager to an external DMS.

DocManager is distributed with a filesystem implementation of this external DMS integration, where the filesystem is used as the DocManagerstorage layer.

Filesystem DocManager Configuration

The beans defined in the `docmgr-external-config.xml` file for DocManager filesystem are as follows.

Bean	Description
`externalDMSConnector`	Main bean, which instantiates `com.ipicorp.docmgr.external.impl.FileSystemImpl`
`performanceMonitor`	Optional performance monitoring bean, which instantiates `com.ipicorp.tools.aop.PerformanceMonitor`

The `externalDMSConnector` bean has the following properties; all defined using standard property Spring nodes.

Attribute	Required/ Optional	Description	Possible Values
rootDir	Required	DocManager root storage directory on the filesystem.	String representing the fully qualified path or the directory
versionPrefix	Required	Prefix used for naming document versions.	Character, such as V
storeLinks		Boolean flag set to true in order to store links on the filesystem.	`True` `False`
namingScheme	Required	Specifies how DocManager constructs the path for a resource which it passes down to	`index0` (default) - the name of the resource is used (such as, `Library Root\Templates\aTemp late.docx`) id - the resource id is used (such as, 0\100\101) entity - the entity is read from an index determined by the `file-`

		the external DMS (if there is one). For more information, see Custom File Path.	`storage-index` attribute in the corresponding entity type in `entity-types.xml` ,
forbidden-characters	Required	Defines the characters to be replaced. > Characters may need to be XML-escaped.	String; space delimited list
replace-forbidden-characters	Required	Defines the characters that replace the characters defined by forbidden-characters. The number of characters defined must be equal to the number of characters defined by forbidden-characters.	String; space delimited list

The `namingScheme` attribute makes it possible to bypass certain OS limitations when storing files on a file system as opposed to RDBMS. The file path for the document resource can be constructed by either using the display names of folders, the internal resource ID of folders, or a combination of the two, including the option to skip entire levels.

Filepath using Folder Names

When setting the attribute `docmgr-filesystem-mapping` with a value of index0, the file path created will use the title index fields from the `entitity-type.xml`.The created path on the file system will look like the following.

```
\images\Docmgr\Entity\Counterparty_A\Product\Trade_ID\doc_V1.EXT
```

Filepath using Resource ID's

When setting the attribute namingScheme with a value of id, the file path created will use the internal resource ID's (from `DOCMGR_RESOURCE` table) of the index fields from `entity-type.xml`. The created path on the file system will look like the following.

```
\images\Docmgr\200\209\210\211\212_V1.EXT
```

Filepath using Folder Names and Resource IDs Combined

When setting the attribute `namingScheme` with to value of entity, an optional entity model element attribute, `file-storage-identifier`, can be used to cause the path element for the given entity to use an arbitrary DocManager index for the pathname (not just index 0).

The following are valid values for `file-storage-identifier`.

- **-2.** Specifies to not include this resource in the path
- **-1.** Specifies to use the resource's numeric ResourceID as the path element
- **Non-negative integer.** Specifies to use the named resource index variable's value as the path element. Characters in that value which cannot be represented in a filename are changed to "_" (underscore) by default. This behavior can be configured in `docmgr-config.xml`.

If no `file-storage-identifier` attribute is specified, the default is 0 (use the index0 value).

For example, if the entity model is:

Entity	file-storage-identifier
Counterparty Id	0
Counterparty Long Name	0
Trade Ref	0
Document	0

Then the pathname used by DocManager would be similar to:

```
images\docmgr\BANKFOO\Bank of Foo\TRADE123\Confirmation_v1.doc
```

The `file-storage-identifier` attribute can be used to eliminate the redundant "Long Name", and to use a numeric resource ID for the document. The following is an example of a possible `file-storage-identifier` setup for the relevant `entity-types`.

Entity	file-storage-identifier
Counterparty Id	0
Counterparty Long Name	0
Trade Ref	0
Document	-1

The resultant paths will then be similar to:

```
images\docmgr\BANKFOO\TRADE123\9936363_v1.doc
```

Custom Indexing Forms

The forms used to edit indexing information in DocManager can be customized at the entity level. Each entity type can be linked to a custom index form for folders and a separate custom form for documents and hyperlinks. If no custom forms are specified, the default indexing form is used.

To add a custom form to DocManager

1. Create a custom JSP page.
2. Place the custom JSP page in the `docmgr-custom- web/customJSPs/jsp/include` directory of your Scrittura/DocManager custom files.
3. Add the name of the custom JSP page to its entity type in the `entity- types.xml` configuration file.

   If the JSP file is not placed in the `/jsp/include` directory, you must also include the full path.

   Example

   An entity type in an entity-types.xml configuration file with custom indexing JSP pages.

```
<entity-type name="Product"
title-index="2" doc-title-index="9"
inherit-parent="true"
doc-form="documentform.jsp" folder-form="folderform.jsp">
<index idx="0"
label="Title" index-type="folder" required="true" type="String"/>
<index idx="2"
label="Product Type" index-type="folder"
required="true" type="String"/>
...
```

Field Validation

Field Validation is a configurable component of DocManager. The default implementation includes validation classes for different field types: `Integer, String, Date, Year, Quarter, SSN,` and `Zipcode`. Index and field values are assigned data types in the `entity-types.xml` configuration file. The DocManager API then validates those index and field values against the associated validation class each time they are saved.

Additional validation classes can be added to the DocManager API to perform custom field validation by completing the following.

1. Develop a new field validation class and add it to the API.
2. List the new validation class as a `<field type>` in the `docmgr- config.xml` file.

Validation Classes

DocManager includes classes to perform validation on the following data types:

- **String**. Accepts any string of characters.
- **Integer**. Checks that the value defined is a signed decimal integer. The first character can be a minus sign, but all the others must be decimal digits.
- **Date.** Accepts the value if it conforms to one of the following formats, and then reformats it to the date-format specified in **docmgr-config.xml**.

MM/dd/yy	MM-dd-yy	dd.MM.yy
MMM dd, yy	yy yyyyMMdd	yy MM dd
dd-MMM-yy	dd/MMM/yy	MMM/dd/yy
dd MMM yy dd	dd MMMMM yy	MMMMM dd, yy

- **Year.** Checks that the value entered is a non-negative integer. If the number entered is between zero and 50, it becomes a 21st century year. If the number is between 51 and 99, it becomes a 20th century year.
- **Quarter.** Accepts one of the following four values, ignoring case: Q1, Q2, Q3, Q4
- **SSN.** Accepts 9 (999999999) or 11(999-99-9999) character Social Security Numbers (SSN), and reformats them to 11 character SSNs.
- **Zipcode.** Accepts 5 digit or 9 digit (zip + 4) zip codes. Zip + 4 values can be either 999999999 or 99999-9999 formats. The 999999999 format is reformatted to 99999-9999.

Field Validation Configuration

Develop a new field validation class that implements the `com.ipicorp.docmgr.validation`.FieldValidation interface and add it to the `com.ipicorp.docmgr.validation` package.

The new field validation class then needs to be added as a new <field- type> node in the `docmgr-config.xml` file.

The following node is for the String validation class:

```
<field-type type-name="String"

validation-class="com.ipicorp.docmgr

.validation.StringValidator"/>
```

The field type is assigned a unique name in the `type-name` attribute. This type-name is then the name that is used in the `entity-types.xml` file to designate an index or field value as being of that type:

```
<index idx="0"
```

```
label="Title"

 index-type="folder"

required="true"

type="String"/>
```

Import Daemon

The DocManager Import Daemon is used to monitor directories for files to be published into DocManager.

The Import Daemon can be configured to monitor directories using the following methods.

1. To look for individual documents in a directory.

   These are documents that have been scanned or faxed and have no metadata associated with them. The Import Daemon publishes these documents into a hidden folder in DocManager, which can then be indexed and published to the correct place in DocManager using the 'Index' functionality of the DocManager web interface.

2. To look for XML meta-data files.

   When there exists a number of documents with known indexing data (such as when a customer needs to perform a migration of documents off an old system into DocManager) one of these .xml metadata files can be produced for each document and placed in the directory to be monitored. The Import Daemon uses these XML files to try to publish the document to the correct place in DocManager. The original document on the filesystem is deleted after publishing.

Note the following about the DocManager Import Daemon:

- The directories to be monitored are configured in the docmgr- `config.xml` file.
- It can be configured to monitor multiple directories.
- It uses a simple workflow that can be changed or expanded as the need arises.

Import Daemon Configuration

The DocManager Import Daemon can be configured as needed to monitor directories for files to be published into DocManager.

To configure the Import Daemon

1. For each directory to be monitored, add an import-dir element to the `docmgr-config.xml` (see `docmgr-config.xml, on page 130`). If no `import-dir` entries are included, the Import Daemon will start up, see that it has nothing to monitor, and stop running. The following example has two import directories configured. The first looks for XML meta-files, the second looks for documents:

   ```
   <docmgr-config use-entity-types="true" security-factory=
   ```

```
"com.ipicorp.docmgr.security.BaseAclFactory" libname="testlib"
disable-security="false" superuser="admin" database="sqlserver">
<import-dir dir="C:\import\monitor_queue_1" use-xml-metafiles="true"
sleep-seconds="10" />
<import-dir dir="C:\import\monitor_queue_2" use-xml-metafiles="false"
sleep-seconds="10" />
</docmgr-config>
```

2. Ensure that the following entity type is in the entity-types.xml configuration file. If it is not present, add it:

```
<entity-type name="Batches"
title-index="fixed" doc-title-index="9" inherit-parent="false">
<index idx="0"
label="Title" index-type="folder" required="true" type="String"/>
<index idx="9"
label="DocTitle" index-type="document" required="true" type="String"/>
</entity-type>
```

3. Ensure that the workflow folder contains a workflow called `DocMgr.xml` (or the workflow-name that is configured for the import directory) and that it is referenced in the `workflow.xml` as well.
4. Run `SetConfig` to activate the Import Daemon.

Import XML Metadata Files

The Import Daemon takes the XML metadata files and attempts to publish the associated document to the correct folder in DocManager.

This file can be laid out as a hierarchy of folders (mapping the entity-type model) plus the document resource, or the file can just include a single document resource entry if all the index data is included that is required to find or create that document's parent path. Versions can be specified by naming one of the indexes or fields 'Doc Version' and specifying a number.

Example: Use a hierarchy to map the entity-type model

```
<resource-op
new-file="/opt/scrittura4/files/sampleFiles/TestDocument.pdf" op="create">
<folder-resource entity-type="Counterparty" visible="true">
<index idx="1" label="Counterparty">Bank</index>
<field idx="0" label="Address">100 Broadway</field>
<field idx="1" label="Phone">555-5555</field>
<field idx="2" label="Primary Contact">Bill</field>
```

```
<folder-resource entity-type="Product" visible="true">
<index idx="3" label="Product Type">Option</index>
<folder-resource entity-type="Deal" visible="true">
<index idx="4" label="Ref Id">300</index>
<index idx="5" label="Counterparty Ref">57</index>
<document-resource entity-type="Deal"
visible="true">
<index idx="8" label="Doc Date"> 08/08/2014
</index>
<index idx="9" label="Doc Type"> Batch Imported Document
</index>
</document-resource>
</folder-resource>
</folder-resource>
</folder-resource>
</resource-op>
```

where

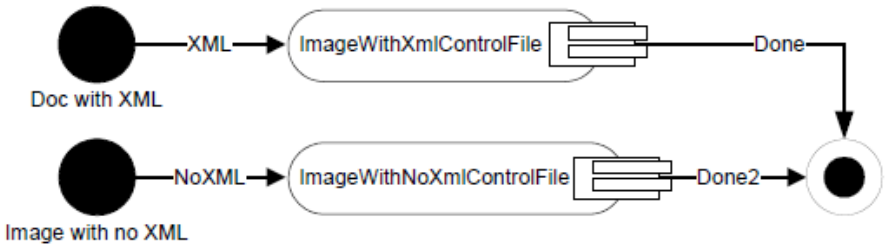
- `new-file` is the full path to the document to migrate
- op is the operation to perform ("create" is currently the only operation supported)

Be sure to include all the required indexes and fields for each resource in the hierarchy.

In each case, include the index or field number (using the `idx` attribute), the label, and the value. Also ensure the values are valid types or the import will not succeed.

Use of the DocMgr.vsd Visio File

The default workflow is called `DocMgr` and the following illustration represents its `.vsd` file



When the import directories have been configured in `docmgr-config.xml`, the workflow activity triggered by the file being dropped into the workflow process depends on the value of the attribute `use-xml-metafiles`.

If `use-xml-metafiles` is set to true, the entry point in the workflow will be “Doc with XML”.In this case, this is an xml control file that is dropped and it is processed by the `ImageWithXmlControlFile` class tool.

If `use-xml-metafiles` is set to false, the entry point in the workflow will be "`Image with no XML`". In this case, the document itself is dropped in the import directory and it is processed by the `ImageWithNoXmlControlFile` class tool.

Audit Trail

DocManager does not have a GUI showing the audit trails like there is in the Scrittura application (history view). However, the following DocManager functions are audited and stored in the audit tables.

- folder creation
- document creation
- hyperlink creation
- new version creation
- resource deletion
- file version update
- hyperlink URL update
- update of a resource's metadata (index/field data)

The audit table columns contain the following information for DocManager audit entries.

- `ACTIVITY`: Create, update, or remove.
- `ACTIONTYPE`: Currently set to "user" for all DocManager entries.
- `ACTION`: This is a message describing the action taken.
- `REFTYPE`: Set to `RESID` for all DocManager entries.
- `RESID`: The DocManager Resource ID.
- `REFSEQ`: This is not currently used in DocManager auditing. This is used in Scrittura to group a series of related actions (all related actions are given the same `REFSEQ` number).

Example 1 - Create a child folder:

```
"Created Child Folder: [folder title] in [resource id of parent folder] Entity=
[entity type of the folder created]"
```

```
"Created Child Folder: Swap in 4801 Entity=Product"
```

Example 2 - Update metadata for a resource:

```
"Updated data"
```

Example 3 - Update a file version:

```
"Updated Datafile Version: [version #] bytes= [file size]"
```

```
"Updated Datafile Version: 1 bytes=142800"
```

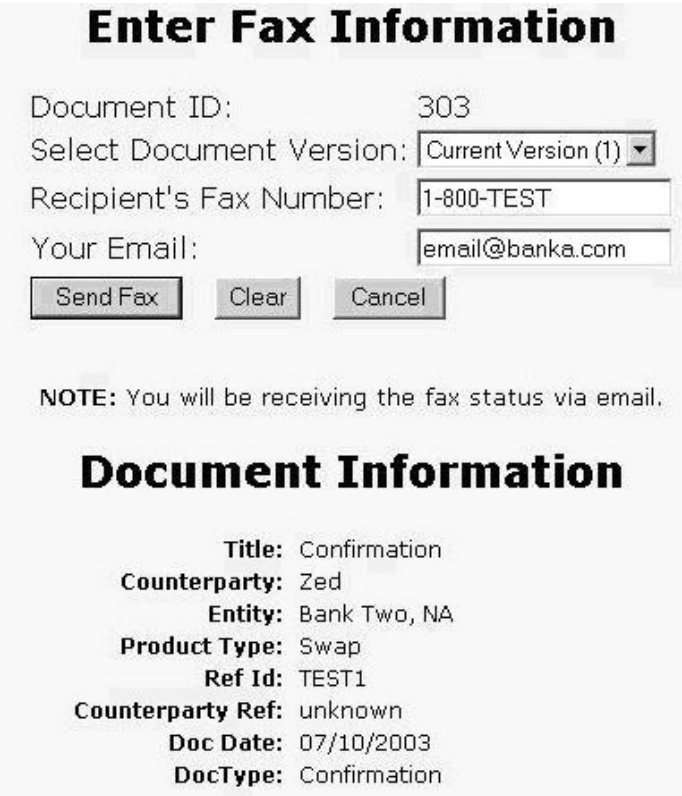
DocManager Faxing and Email

DocManager provides the framework to fax documents.

- Two built-in DocManager faxing JSPs and two email JSPs are provided and can be customized for your specific faxing and email needs. These JSPs provide a GUI for fax input (such as fax number, email, and so on) and a post-fax status message.
- Interface methods that all custom DocManager faxing and email implementations must implement.

GUI Method for Faxing and Emailing

The `faxInput.jsp` and `faxResults.jsp` files, both located under jsp/include, provide the user interface for DocManager faxing. The `faxInput.jsp` outputs a GUI, letting the user define the document version, recipient's fax number, and his/her email. The `faxInput.jsp` can be customized if there is additional required faxing information. Similarly, `emailInputMainSection.jsp` and `emailResultMainSection.jsp`, also located under `jsp/include`, are provided for emailing.



After sending the fax, the `faxResults.jsp` displays a status message on whether the fax was sent to the Fax Server:

> This status message is NOT the status of the fax. It is a status message describing whether the document was successfully sent to the Fax Server to be faxed to the recipient.

Faxing and Emailing Configuration

All DocManager faxing implementations must implement the `faxDocument` method in the `com.ipicorp.docmgr.fax.DocManagerFaxer` interface.

```
package com.ipicorp.docmgr.fax;

import javax.servlet.http.HttpServletRequest; public interface DocManagerFaxer

{

public void faxDocument(HttpServletRequest request)

throws Exception;

}
```

There is a similar interface for emailing:

`com.ipicorp.docmgr.fax.DocManagerEmail.`

DocManager faxing and emailing are configured in docmgr-config.xml by adding the `<fax>` and `<email>` nodes. For more information on these nodes, see <fax> node, on page 134 and <email> node, on page 134.

```
<fax>

<fax-enabled>true</fax-enabled>

<fax-class>com.ipicorp.docmgr.banka.SampleDocmgrFaxer</fax- class>

</fax>

<email>

<email-enabled>true</email-enabled>

<email-class>

com.ipicorp.docmgr.banka.SampleDocmgrEmailer

</email-class>

</email>
```

When faxing and emailing are enabled, Fax Document and Email Document links display in the user interface resource menu, provided that the user has the appropriate permissions.

DocManager API

This section includes information about DocManager of code usage. A basic description of the interfaces is provided as well as examples of the processes for creating resources and adding versions. For other operations, see the Javadoc provided with the Scrittura release.

For detailed information about the DocManager wrapper API, see Appendix D: DocManager Wrapper API, on page 424.

DocManager Interface Configuration Location

All of the DocManager interface classes are located in the `com.ipicorp.docmgr.docmgrinterface` package. The `ResourceData`, `FileData`, and various DocManager exception classes are located in `com.ipicorp.docmg.util`. These packages are all located within the `ipidocmgr.jar` located in the `\lib` folder of the Scrittura distribution.

How to Call the Resource Interfaces

DocManager resources operations are performed using methods in the following interfaces.

- Generic functionality is available through the

  `DocmgrResourceInterface.`
- Resource-type-specific functionality is available using the resource-specific interfaces:
  - `DocmgrFolderInterface`
  - `DocmgrDocumentInterface`
  - `DocmgrLinkInterface`

Each of these interfaces extends the DocmgrResourceInterface. Implementations of the interfaces are accessed using the DocmgrInterfaceFactory.

Example

```
DocmgrInterfaceFactory docmgrInterfaceFactory
= new DocmgrInterfaceFactory();
DocmgrDocumentInterface docmgrDocumentInterface
= docmgrInterfaceFactory.getLocalDocumentInterface();
```

API Example: Document Creation and Version Addition

The following is a DocManager API code example for document creation and version addition.

```
final DocmgrInterfaceFactory docmgrInterfaceFactory
= new DocmgrInterfaceFactory();
final DocmgrDocumentInterface docmgrDocumentInterface
= docmgrInterfaceFactory.getLocalDocumentInterface();
final FileData fileData = new FileData();
fileData.setData(fileContent); fileData.setExtension(extension); fileData.setIndices
(indices); fileData.setFields(fields); fileData.setFullTextSearchEnabled(true);
final ResourceData data
= documentInterface.get(parentId, fileName);
int id = (data == null)
```

```
? documentInterface.create(parentId, fileName, fileData)

: documentInterface.addVersion(data.getId(), fileData).getId();
```

API Example: Folder Creation

The following is a DocManager API code example for folder creation.

```
final ResourceData data = new ResourceData(); data.setIndices(indices);
data.setFields(fields);

docmgrFolderInterface

.create(parentId, folderName, entityTypeName, data);
```

Chapter 6: User Interface Configuration

This section provides details about the internals of the Scrittura user interface and the tools available to configure and customize it.

This section contains the following topics:

Scrittura MVC Model

All requests made by Scrittura JSP pages use the Model View Controller (MVC), which mediates the JSP views and the data in the database.

As a result, any custom JSP pages that require access to the Product Instance variables must use this controller.

Custom JSP Pages

One Scrittura JSP page can contain a link to another JSP page that can also display the same Product Instance (PI) variables.

Such links must be built using the controller with the event (e) with a value of "PI". The JSP code should appear in a single line.

Exmple: Link a JSP Page to another JSP Page

```
<a target="_new" href="controller?e=pi&v=Variables

&p=<scrittura:value name="CommonReferenceID"/>" > Product Variables

</a>
```

This code creates a hyperlink titled "Product Variables" in the JSP page. It creates a new window (target="_new") defined by the view configured as Variables (parameter "v in the URL). The `CommonReferenceID` value is passed (using the `scrittura:value` JSP tag) to identify the particular Product Instance variables to be viewed (the "p" parameter in the URL).

The view defined as "Variables" must be defined as a view in the `scrittura-config.xml` page. For example:

```
<view name="Variables"

type="custom"

view="" script=""

frameset="/AllProductInstanceVars.jsp"/>
```

The linked JSP page as defined above can make use of the `<scrittura:edit name="SaveAndClose"/>` tag to create a Save button if the linked page includes `<scrittura:edit>` widgets.

If the hyperlink to the new page references the Product Instance using the "i" parameter in the URL instead of the "p" parameter, the tag above creates a **Save** and **Forward** button.

Validate View Data

To perform server-side validation of views, you can define validation classes using the `validator-class` attribute in the `<view>` element in the `scrittura-config.xml` file. Any validation class should implement the `interface com.ipicorp.scrittura.web.events.SaveEventValidator` as follows:

```
public interface SaveEventValidator

{

public Map validateEvent(HttpServletRequest request,

Map piVariables)

throws ValidationException;

}
```

NOTE: Server side validation of views is possible by assigning a validation class to a particular view.

Add Custom Events

In keeping with the MVC model, you can add custom events to the general Scrittura controller.

To add custom events

1. Create a custom event by extending the `com.ipicorp.mvc.EventHandlerBase` class.
2. Edit the `mvc-extend.cfg` file. If there are no custom events, the `mvc- extend.cfg` file may be empty, but it still must exist.
3. Add each custom event in the following format:

   ```
   yourEventName, yourEventClass
   ```

   Example:

   myEventName, com.ipicorp.scrittura.web.events.MyEvent
4. Call the event from JSPs with forms or links such as:

   ```
   <a href="controller?e=myEvent...>Run My Event</a>
   ```

NOTE: Once the system has been rebuilt and restarted, the Scrittura configuration must be reloaded using the usual SetConfig process (see Scrittura Administration and Run-Time, on page 369).

Configure Application Appearance

This section details the basic ways to configure the appearance of the Scrittura application (such as, queues). Additional configuration levels are also available and discussed elsewhere in this guide (such as, bulk screens in Queue Screen Configurations, on page 188).

Customize Queue Columns

The `scrittura-config.xml` file controls the appearance and links available to users when they open an activity worklist, including the columns displayed.

For example, a worklist (queue element) might appear in this file as:

```
<queue name="First Signature" activity="Scrittura.Signature_A">
<column display="Counterparty" variable="Party[B]"/>
<column display="Product Type" variable="ProductDefDisplay"/>
<column variable="TradeDate"/>
<column variable="Trader"/>
<column variable="TradeAmended"/>
<view name="Sign"
type="view" view="/signature_a.jsp"/>
<view name="Tracking" type="custom" view="/tracking.jsp" script="trackingRouter.bsh"
frameset="/trackingFrame.jsp"/>
<role name="signers_a"/>
</queue>
```

To customize columns

1. In the `scrittura-config.xml` file, add, or re-order `<column>` elements.
2. Specify (potentially) separate display strings and variable names.
3. After changes to the `scrittura-config.xml` file, reload the file from the `SetConfig` screen.

Alternatively, it is possible to define column sets in `scrittura-config.xml` and use those column sets within the queue elements. This lets you easily reuse the same set of columns for different queues as needed.

Display the Latest Annotation in the Queue Screen

When specifying the column information in the queue definition in `scrittura-config`, if a queue has a name prefixed by "annotation" it displays the latest annotations from the annotation thread with this name (`annotation_info` will display the latest entries in the info thread).

Annotation display can be customized as follows:

- Use the following convention: `_annotation_x_` followed by the thread name, where "x" is the number of annotations to display (for example, `_annotation_5_info`) to display. The default number of annotations is 10.
- Add nohdr to the prefix if Scrittura should not display the user name and date of the annotation (`_annotation_5_nohdr_info`).

Define Worklist Hyperlinks to JSP Views

In the following worklist (queue element) example, the queue definition defines two hyperlinks.

```
<queue name="First Signature" activity="Scrittura.Signature_A">
<column display="Counterparty" variable="Party[B]"/>
<column display="Product Type" variable="ProductDefDisplay"/>
<column variable="TradeDate"/> <column variable="Trader"/>
<column variable="TradeAmended"/>
<view name="Sign"
type="view" view="/signature_a.jsp"/>
<view name="Tracking" type="custom" view="/tracking.jsp" script="trackingRouter.bsh"
frameset="/trackingFrame.jsp"/>
<role name="signers_a"/>
</queue>
```

- One "`view`" type view, "Sign", whose view is defined by `signature_a.jsp`.
- A "`custom`" type view,"Tracking", whose view is defined by tracking.jsp, and is using trackingFrame.jsp as its main frame. The script is run when the user clicks `Save` or `Save & Forward` buttons on the page. This script can be used to update dependent values (such as if you only want to show the fax number for a fax, or if you change legtype[A] from fixed to float to enable a whole new set of inputs). It can also do anything you can do in a BeanShell script.

Other view types supported include Edit and Bulk. A `History` link is also included by default.

The `<role>` defines those user roles with permission to view documents in the queue. Conditions may be added to the `<role>` element to narrow the set of workflow items available to a user.

> **NOTE:** After making changes to the scrittura-config.xml file, reload the file from the SetConfig screen.

JSP View Requirements

All JSP files must include the following rows at the beginning of the page:

```
<%@ page language="java" import="java.lang.*" %>
<%@ taglib uri="scrittura3.tld" prefix="scrittura" %>
```

When configuring JSP files, you can only define a form once. This means that if your view defined in `scrittura-config.xml`, `"view="/jsp/review.jsp"` has the following form elements, they should not be present in any other JSP files included by your view JSP:

```
<scrittura:edit widget="hidden" name="ProductInstanceVersion" />
<scrittura:edit widget="hidden" name="i" />
<scrittura:edit widget="hidden" name="e" />
<scrittura:edit widget="hidden" name="u" />
```

Use Style Sheets to Customize Application

Scrittura uses a number of style sheets to control the appearance of various aspects of the application.

The following style sheets are included in an initial deployment.

- main.css
- mainPage.css
- menu.css
- queue.css
- SetConfig.css
- sortableTableStyle.css

In addition, JSP files, Product Definition views, and document templates may be customized or created to invoke entirely new style sheets.

When linked in a JSP file or template, they would be referenced with a path similar to that shown here:

```
<link type="text/css" href="/scrittura/stylesheets/docmgr.css">
```

Change Framesets to Arrange JSP Views

Scrittura comes with standard framed window for pages of type view, which offers a default layout. A view of type View shows the document on the right, trade details on the left, and annotations on the bottom. A view of type Edit shows these elements in a different layout.

The following framesets are provided with the Scrittura core distribution.

Frameset	Description
/jsp/viewframe.jsp	Frameset that provides view, doc, and annotate frames, respectively for trade details, document, and annotations.
/jsp/viewFrameset.jsp	Similarly to the `viewframe.jsp` frameset, this frameset provides view, doc, and annotate frames. In addition it also provides a docedit frame in order to display document edition option on top of the document frame. The content of the `docedit` frame is defined by the custom DocControls view element defined in `scrittura-config.xml` for that queue.

To customize the display for elements in a queue, the queue itself must be defined as a `"Custom"` type in the `scrittura-config.xml` file.

In the following example, the `"Exceptions"` queue defines a link titled `"Open"` which leads to a frameset defined by the file `customFrame.jsp`. This definition also executes the trackingRouter BeanShell script each time a user opens a document in the queue.

```
<queue name="Exceptions" activity="Scrittura.Exceptions">

<column display="Counterparty" variable="Party[B]"/>

<column display="Product Type" variable="ProductDefDisplay"/>

<column variable="TradeDate"/>

<column variable="Trader"/>

<column variable="TradeAmended"/>

<view name="Open"

type="custom" view="/custom.jsp" script="trackingRouter.bsh"
frameset="/customFrame.jsp"/>

</queue>
```

This particular view could also be defined, using conditions, to apply only to a particular user role.

Set Preferences for Date and Currency Formats

User preferences can be set using the userprefs.jsp JSP page. It allows setting the preferred date format as well as decimal and thousand separators for currency amounts.

The "Date Input Format" defines the format a user must use when typing a value for a 'date' variable. The "Date Output Format" defines the way these dates are presented on screen in queue lists and other JSP views.

The following are examples of valid formats and corresponding sample entries.

Format	Example
MM/dd/yyyy	06/15/2013
d-MMMM-yy	15-June-13
dd/MMMM/yyyy	15/June/2013
MMMM d, yyyy	June 15, 2013

Configure User Preferences

The preference behavior is determined by the class:

`com.ipicorp.scrittura.util.IpiUserPreferenceManager`

This class must be named in the `<scrittura-config>` element of `scrittura-config.xml`:

`user-preference-manager`

```
="com.ipicorp.scrittura.util.IpiUserPreferenceManager"
```

The User Preferences screen is rendered by the following hyperlink, defined in the built-in JSP file `menu.jsp`.

```
/scrittura/controller?e=userprefs
```

Global, User, and Document Format Preferences

Global format settings are determined directly in the `<scrittura-config>`element of `scrittura-config.xml`:

```
separator="," decimal="."
```

```
date-format="yyyy-MM-dd"
```

```
date-parse-format="yyyy-MM-dd"
```

These settings determine the default formats for document generation and views for each user who has not declared their own set of preferences.

Documents generated from HTML or JSP templates will follow these global settings unless they are specifically overridden with changes to the following variables on the Product Instance (by Beanshell or other means):

- `_pi_DateFormat`
- `_pi_Separator`
- `_pi_Decimal`

For example, a Beanshell script called in the workflow could change formats for a particular class of client:

```
if (clientLocale.equals("EUR"))
```

```
{
```

```
_pi_DateFormat = "dd-MMM-yyyy";
```

```
}
```

Admin User Format Preferences

Admin users should not customize their own display preferences as these preferences will trickle down to all other users.

JSP Tag Library Reference

This section describes the contents of the Scrittura JSP tag library.

Using the JSP Tag Library

The Scrittura tag library contains several custom tags. These tags are used extensively in the views.

To use the library, add the following to your JSP:

```
<%@ taglib uri="scrittura3.tld" prefix="scrittura" %>
```

All the tags support an optional "class" parameter that sets the CSS class of the generated elements.

> **NOTE:** It is also possible to add custom tag libraries. For this purpose, copy the tag library `.tld` and `potentially .jar` files in the Scrittura source directory under `/custom-web/customTagLibs`.

Edit Tag

The Edit tag generates an edit control for a product variable.

```
<scrittura:edit name="variable"

[widget=""]

[rows="#"]

[cols="#"]

[class=""] [excludes=" , , "]

[required="true|false"]/>
```

"Widget" corresponds to the list of widgets in the Product Definition. If no widget is specified, then the widget defined in the Product Definition is used. If no widget is defined in the Product Definition, then a text-box is used.

When enclosed in a `<scrittura:foreach>` tag, array indexes are specified using `"[ ]"`, for example `"Party[ ]"`. If enclosed within two `<scrittura:foreach>` tags, array indexes are specified using `"[ ][ ]"`, for example. `"PartyContact[ ][ ]".`

The `<scrittura:edit>` tag includes an excludes optional attribute. It goes last in the list, after the cols optional attribute. The value is a comma delimited list of values to be removed from a drop-down or radio button list.

These values would not refer to the ordinal numbers of possible choices, such as `"excludes=4,5,6"`. These values are the actual strings that would otherwise appear in the drop-down, such as "Trader" or "Finished".

The `excludes` attribute is also exposed in the Product Definition. `<VariableEditSpec>` has the optional attribute `excludes-list`. This list is copied directly into the JSP and is only relevant if the variable is displayed using a drop-down or radio list.

The required attribute (true or false) determines whether or not the user must enter a value in order to successfully forward a workitem out of a manual activity. Blank values are rejected.

Scrittura supports a drop-down list of available images. For example, the following JSP code will list the titles of all of the Image documents in all of the Image folders in DocManager.

```
<scrittura:edit name="myImage" widget="dropDownList"/>
```

Scrittura adds enhancements to the **SaveAndClose** submit buttons (**Save & Fwd**, **Save**, **Save & Close**, and **Close** buttons).

- To call a JavaScript function on a submit button, use the following attributes:
  - `savefwdjscript`
  - `savejscript`
  - `saveclosejscript`
  - `closejscript`

  The following example calls a JavaScript function when the Save &Fwd button is pressed:

  ```
  <scrittura:edit name='SaveAndClose'
  savefwdjscript="myJavaScriptFunction()" cols='4' >
  Save Buttons
  </scrittura:edit>
  ```

- To show a different display value on a submit button (for example, `Save & Forward` instead of the `Save & Fwd` default), use the following attributes:
  - `savefwdvalue`
  - `savevalue`
  - `saveclosevalue`
  - `closevalue`

  Other areas of Scrittura are expecting the default display values (`Save & Fwd`, `Save`, `Save & Close`, `Close`) when the form is submitted. To work around this, use JavaScript to set the value of the `SaveAndClose` variable to one of the default values before the form submits

  The following example displays a **Save & Forward** button, instead of the `Save & Fwd` default:

  ```
  <scrittura:edit name='SaveAndClose'
  savefwdvalue="Save & Forward" cols='4' >
  Save Buttons
  </scrittura:edit>
  ```

- To disable a button, use the following attributes:
  - `savefwddisable`
  - `savedisable`
  - `saveclosedisable`
  - `closedisable`

  The following example disables the Save button:

  ```
  <scrittura:edit name='SaveAndClose'
  savedisable="true" cols='4' >
  Save Buttons
  ```

```
</scrittura:edit>
```

- To rename the `SaveAndClose` input variables, use the following attribute:
  - `saveandclosename`

  The following example renames the default SaveAndClose input to `submitButtons`:

  ```
  <scrittura:edit name='SaveAndClose'
  saveandclosename="submitButtons" cols='4' >
  Save Buttons
  </scrittura:edit>
  ```

Value Tag

The Value tag outputs the formatted text of variable (or, in the case of Image variables, the image itself).

```
<scrittura:value [class="style"]
[changeclass="style"] [escapespecialchars="true | false"] name="variable" />
```

where,

- `class` is the style name for unmodified variables
- `changeclass` is the style name for modified variables.
- `escapespecialchars` determines whether the tag will escape special characters like <, >, ", or & for the outputted text. This parameter is `false` by default.

  When enclosed in a `<scrittura:foreach>` tag, array indexes are specified using `"[ ]"`, for example `"Party[ ]"`. If enclosed within two `<scrittura:foreach>` tags, array indexes are specified using `"[ ][ ]"`, for example. `"PartyContact[ ][ ]"`.

  Scrittura creates generated output such that the value is displayed using the `<td.valueCell>` class from the style sheet.

  Scrittura adds support for image variables. For example, to display the image document in 25 by 25 pixel area, the value tags would be formatted as:

  ```
  <scrittura:value name="myImage" valuetype="Image" imageheight="25"
  imagewidth="25" />
  ```

  To compress and display the document to half of its natural size, for example, the value tags would be formatted as:

  ```
  <scrittura:value name=" myImage " valuetype="Image" imageheight="50%"
  imagewidth="50%" />
  ```

Label Tag

The Label tag outputs the formatted text label of variable.

```
<scrittura:label name="variable" [class=""] />
```

where,

- `name` is the display name of a variable defined in a Product Definition.
- `class` is the style name for the label display.

When enclosed in a `<scrittura:foreach>` tag, array indexes are specified using "`[ ]`", for example "`Party[ ]`". If enclosed within two

<scrittura:foreach> tags, array indexes are specified using "`[ ][ ]`", for example. "`PartyContact[ ] [ ]`".

DocManager Document Tag

This tag creates a table with links to each of the documents in the DocManager folder for the Product Instance on the current page.

```
<scrittura:dmsdocs style="dms"
[indices=""]
[fields=""]
[changedefaultdoc=""]
[defaultdocroles=""]/>
```

where,

- `style` must be dms. dms is currently the only supported value
- `indices` defines the value for each of the listed index fields to be added to the display. Multiple index fields must be separated by a space.
- `fields` adds to the display, for each document, the value for each of the listed field indexes. Multiple field indexes must be separated by a space.
- `changedefaultdoc`, optional, specifies whether to display a radio button in front of each document resource. The default is false, meaning that the radio button does not display. The selected document is the default document resource for the current Product Instance. By clicking manually on another radio button, one can change the default document. Further document generation steps in the workflow might change the default document again and overwrite the manual selection.
- `defaultdocroles`, optional, defines the user role that has access to the radio buttons in front of document resource and can manually change the default document. Supported values are "" or a valid role.

These indexes and index fields are configured in the `entity-types.xml` file.

For example, if `entity-types.xml` includes the following configuration:

```
<entity-type name="Counterparty"
title-index="1" doc-title-index="9"
inherit-parent="false">
<index idx="0" label="Title"
index-type="folder" required="true" type="String"/>
```

```
<index idx="1" label="Counterparty"
index-type="folder" required="true" type="String"/>
<index idx="8" label="Doc Date"
index-type="document" required="false" type="Date"/>
<index idx="9" label="DocType"
index-type="document" required="true" type="String"/>
<field idx="0" label="Address"
field-type="folder" required="false" type="String"/>
<field idx="1" label="Phone"
field-type="folder" required="false" type="String"/>
<field idx="2" label="Primary Contact"
field-type="folder" required="false" type="String"/>
...
</entity-type>
```

then the following dmsdocs JSP tag creates, in the display, a hyperlink containing the title and version number of each document for the Product Instance, along with each document date, a phone number, and a primary contact name.

```
<scrittura:dmsdocs style="dms" indices="8" fields="1 2"/>
```

Conditional Tag

The conditional tag is used to evaluate conditions pertaining to variable(s).

```
<scrittura:if condition="BSH Conditional Expression">
....
</scrittura:if>
```

The body is included if the BSH expression evaluates to `true`.

If enclosed with a <scrittuta:foreach> tag (or tags), array indexes in the BSH condition should be expressed using the built-in variables "i1" and "i2". For example "`Party[i1]`".

An example using "if" and "else":

```
<scrittura:if condition='Variable1.equals (\"Applicable\")'>
<TR>
<TD>
Variable Text: <scrittura:value name="Variable1"/>
</TD>
</TR>
```

```
</scrittura:if>

<scrittura:else>

<TR>

<TD> Some other text </TD>

</TR>

</scrittura:else>
```

An example using "if" and NOT equals (which requires an exclamation mark):

```
<scrittura:if condition='!Variable1.equals (\"Applicable\")'>

<TR>

<TD>

Variable 1 is not applicable </TD>

</TR>

</scrittura:if>
```

The condition parser used for the scrittura:if tag is the same as the one used to parse conditional transitions in the workflow. For more information, see Condition Parser, on page 109.

Search List Tag

Search List tag outputs a `<select>` form object with `name="parameter name"` and the value is the list of valid search variables from the Scrittura configuration.

```
<scrittura:searchvars name="parameter name" [class=""] />
```

Iteration Tag

This tag iterates over the enclosed body once for each element of the named variable array.

```
<scrittura:foreach name="> ... </scrittura:foreach>
```

where,

- `name` is the name of the array variable to iterate over.

For a 2D array, enclose a second <scrittura:foreach> tag and specify the name as "`name[ ]`".

For example:

- `<scrittura:foreach name="array">`
- `<scrittura:foreach name="array[ ]">`

Open Link in Window Tag

You can create a hyperlink that opens a new window "target" to the page "href" using the window open options "options".

```
<scrittura:a target=""
```

```
href="" options="" [class=""]>

Label

</scrittura:a>
```

Barcode Tag

Applies to HTML, JSP, and WordML templates. DOCX templates handle barcodes differently.

The barcode tag outputs a barcode in a Scrittura generated document (such as Confirmation).

```
<scrittura:barcode value="" [code=""] [size=""]/>
```

where,

- `value` is the string to be encrypted in the barcode.
- `code` must be "128", "39", or "DataMatrix". "128" and "39" generate a 1D barcode. "DataMatrix" is used for 2D barcode generation. The default value is 128.
- `size` is the size of the barcode, in pixels.

The following is a sample of a 1D barcode:

```
<scrittura:barcode value="stringToEncrypt" code="128"/>

The following is a sample of a 2D barcode:

<scrittura:barcode value="stringToEncrypt"

code="DataMatrix" size="100"/>
```

Preview Tag

Applies to HTML, JSP, and WordML templates. Not supported for DOCX templates.

The preview tag displays a hyperlink in JSP that will allow you to preview the document template "name" for the current Product Instance.

```
<scrittura:preview [class=""] [name=""] />
```

> **NOTE:** The template name has to exist in the Product Definition for that PI in order to work.

`Example`

The tag `<scrittura:preview name="Confirmation">` opens a new browser window and shows the generated document "Confirmation" based on the current data and embedded business logic.

Redline Tag

Redline tag outputs a hyperlink that allows random redlining of all versions of an existing document resource for the current Product Instance. The document and its version have to be generated by Scrittura's document generation engine in order to be eligible for redlining.

```
<scrittura:redline name="" includeFormVars="" [mode=""]> />
```

where,

- `name` is the title of the document (for example, "Confirmation"), being used for redlining. This exact title has to be defined in the corresponding Product Definition as well.
- `includeFormVars` specifies whether the redlining functionality will include any edited, but not yet saved variables in the current version of the document. This attribute should only be set to "true", when redlining is used in a Data Enhancement screen.
- `mode (optional)`, when being set, it's diffing the whole document against each other. If that attribute is not being set, redlining is only evaluating differences in variable values as opposed to the whole document.

Prerequisites

The redline tag requires additional parameters to be configured in the JSP, from where it is being used. The following is a sample for such a JSP:

1. Include the following at the top of the JSP:

   ```
   <%@ page language="java" import="java.lang.*" %>

   <%@ taglib uri="scrittura3.tld" prefix="scrittura" %>
   ```

2. Add the following parameters:

   ```
   <input type="hidden" name="d" value="">

   <input type="hidden" name="dt" value="">

   <input type="hidden" name="includeFormVars" value="">

   <input type="hidden" name="s" value="">
   ```

   after

   ```
   <form method="post" action="controller">

   <scrittura:edit widget="hidden"

   name="ProductInstanceVersion"/>

   <scrittura:edit widget="hidden" name="i"/>

   <scrittura:edit widget="hidden" name="e"/>

   <scrittura:edit widget="hidden" name="u"/>
   ```

3. Declare the redline tag in the JSP with its corresponding attributes and values.

Current Queue Tag

The `currentqueue` tag creates a hyperlink to the document in its current manual queue. If the document is currently in more than one queue, the link points to the most recent queue.

```
<scrittura:currentqueue [class=""]/>
```

History Tag

The `historylink` tag outputs a hyperlink to the current Product Instance's history view.

```
<scrittura:historylink [class=""]/>
```

Loop Tag

The loop tag uses BeanShell to evaluate a condition on the Product Instance attached to the request as "pi". If the condition evaluates to true, the body is included. Otherwise the body is discarded.

```
<scrittura:piloop/>
```

Example:

This code extract shows how to create a table with variable values of multiple PIs in a single document (that is, Multi Chaser).

```
<center>

<table border="1" width="80%" cellspacing="0">

<tr>

<td><b>Our Ref #</b></td>

<td><b>Your Ref #</b></td>

<td><b>Product Type</b></td>

<td><b>Trade Date</b></td>

<td><b>Notional Amount</b></td>

</tr>

<%

String partyB;

partyB = (String) pi.getDisplayValue("Party[B]", false); log.debug("Party[B]=" +
partyB);

ProductInstanceLocalHome piHome = IpiScritturaFactoryUtil.getFactory().

getProductInstanceLocalHome();

List pis = new ArrayList();

for(int i = 0; i < crids.length; i++)

{

pis.add(piHome.findByCommonRefId(crids[i]));

}

log.debug("Results have: " + pis.size()); String piList = "";

%>

<scrittura:piloop>

<tr>

<td><scrittura:value name="CommonReferenceID"/> </td>

<td><scrittura:value name="CounterpartyRefNo"/> </td>
```

```
<td><scrittura:value name="ProductDefDisplay"/> </td>

<td><scrittura:value name="TradeDate"/> </td>

<td><scrittura:value name="Currency[A]"/>

<scrittura:value name="CurrencyAmount[A]"/> </td>

</tr>

</scrittura:piloop>

</table>

</center>
```

File Upload Tag

File Upload tag creates a link to allow users to upload a file to add to the folder or to replace a document for a trade. This tag can be used on any view. After the file is uploaded, the user is returned to the same view.

<scrittura:fileupload title="Manual Confirm"

setdef=""

logic=""

role=""

checkVers="" />

where,

- `title` is the title of the document (such as, "Confirmation"), when being stored in DocManager. If the document resource already exists, it is either being replaced or versioned. If the document does not exist, a new resource is created. A new version of the document is created if the trade version does not match the document version stamp in DocManager.
- `setdef` specifies whether the document will be set as the "Default" document for the trade. If set to "true", the document will be set as the "Default" document for the trade.
- `logic` is the name of a BeanShell script to run after the document has been successfully uploaded.
- `role` defines the role that has access to this function. Supported vales are "" or a valid Scrittura role. If a role is defined, only that role has access. If the user is not in the role, the link is not shown. If the role is "" the upload link is available to all users of the view.
- `checkVers` specifies whether the document will enable versioning of the uploaded documents. Set to "false" to enable versioning of the uploaded document.

Example:

```
<scrittura:fileupload title="Upload Memo"

setdef="false"

logic="setUpload.bsh"

role="signers_a"
```

```
checkVers="false"/>
```

> **NOTE:** This tag is associated with the built-in fileupload.jsp. This JSP can be further customized to have a drop-down list of predefined document titles instead of a free text field.

Preload Variable Tag

The preload variable tag explicitly forces a reload from the database for each variable in the Product Instance.

```
<scrittura:preload />
```

This tag explicitly forces a reload from the database for each variable in the Product Instance.

The `preload` tag should be used in rare cases where a Product Instance variable is changed by the JSP and the new value needs to be displayed. Since the JSP holds the old value of the variable, the preload tag allows loading the new value of the variable.

> **NOTE:** While the name of the tag is "preload", the action is actually a 'reload' of the JSP page.

> **CAUTION:** This technique reloads all of the variables from the database so it is a potentially expensive action and should be avoided.

Counter Tag

The `counter` tag maintains sets of counters throughout a document. Using this tag, you can impose an "outline" structure without explicitly knowing which value is "next".

```
<scrittura:counter type="123|abc|ABC|iii|III" [reset="true"]/>
```

Five counters can be maintained through a document:

- 123 = 1, 2, 3 ...
- abc = a, b, c, ... z, aa, bb, cc, ... zz, aaa ....
- ABC = A, B, C, ...
- ii = i, ii, iii, iv, ...
- III = I, II, III, IV, ...

Use the `reset="true"` specification to reset a given counter. The following is sample JSP code:

```
Section: <scrittura:counter type="123"/> Subsection: <scrittura:counter type="abc"/>

<scrittura:counter type="abc" reset="true"/>

Section: <scrittura:counter type="123"/> Subsection: <scrittura:counter type="abc"/>
```

The above sample produces the following output:

```
Section: 1

Subsection: a

Section: 2
```

```
Subsection: a
```

This tag works in iterations (foreach tags) and over included JSP pages.

Readwrite Tag

The readwrite tag works like the scrittura:if tag, keying off the user's permission inside the queue displaying the JSP page.

Example:

```
<scrittura:readwrite access="write">
```

This is seen by users with write access to this queue

```
</scrittura:readwrite>
```

```
<scrittura:readwrite access="read">
```

This is seen by users with read-only access to this queue

```
</scrittura:readwrite>
```

Signature Tag

The signature tag specifies whether to display a link to access the Signature applet and sign Inbound TIFF documents.

```
<scrittura:signature newwindow="true | false"
source=""
previewSource=""
dest="" image=""
previewImage="" />
```

The `signature` tag has the following parameters.

Parameter	Required/ Optional	Description
newwindow	Optional	The signature tag launches the applet in the same frame by default. Set this parameter to `true` to display the applet in a new window.
source	Optional	DocManager title of the source document to sign. If unspecified, the default PI document will be loaded by the applet.
previewSource	Optional	DocManager title of the document loaded by the applet, which defaults to the source document if unspecified.
dest	Optional	DocManager title of the resulting document once signed. If

		unspecified, this parameter takes the same value as the source document.
image	Optional	Signature image used to sign the document. If unspecified, this parameter defaults to the image associated with the signatory defined by the user name suffixed by “-inbound”, such as “signerA-inbound”.
previewImage	Optional	Image used to preview the signature while signing the confirmation. If unspecified, Scrittura will attempt to take the image associated with the signatory preview-inbound.

Queue Search Tag

The queue search tag lets you to display standard and advanced queue searches in the built-in bulk.jsp and queue.jsp files.

```
<scrittura:queuesearch queuename="queue name" searchtype="qstandard | qadvanced"/>
```

Document Format Tag

The document format tag provides TIFF formatting functionality such as rotation and splits. A "Document Formatting" link appears on the JSP where this tag is applied. The link will not be visible if the document is not of type TIF or TIFF.

```
<scrittura:documentformat [ticketqueue="scrittura_tickets | scrittura_tickets2 |
scrittura_tickets3 | scrittura_tickets4 | scrittura_tickets5"]/>
```

where,

- `ticketqueue` (optional) defines the name of the ticket queue. By default all split TIFF documents are received from the scrittura_tickets JMS queue. If required, other ticket queues may be used by specifying this parameter.

For this tag to work, the "q" parameter must be passed as an argument. This parameter can be set in the frameset of that queue, as shown in the following sample:

Assuming the queue definition in `scrittura-config.xml` contains the following view,

```
<view name="View"

type="custom"

view="/Inbound/Incoming_Documents.jsp"

script=""

frameset="/Inbound/indexframe.jsp"/>
```

The `indexframe.jsp` JSP should then contain the following:

```
<frameset name="docview" cols="50%,*">

<frame

name="view" src="controller?e=view&q=<%=request.getParameter("q")%>

&i=<%=request.getParameter("i")%>
```

```
&v=<%=request.getParameter("v")%>

&crid=<%=request.getParameter("crid")%>" />

<frame name="doc"

src="<%= request.getAttribute("docurl")%>" />

</frameset>
```

General User Interface Configuration

In addition to the edition of JSPs using the Scrittura tag library or the ability to extend the MVC mode, Scrittura also provides flexible configurable options for the user interface, including but not limited to the following.

- Workflow views and queue views
- Configuration of custom panels used in bulk screens and trade detail screens
- Search result pages (“queue style search”)
- Bulk screens
- Trade detail screens

  Configuration of these options is completed by modifying the following configuration files.
- `general-ui-config.xml`
- `scrittura-queue-view-config.xml`
- `scrittura-config.xml`
- `economic-panels-config.xml`

> **NOTE:** If these files are not modified, existing user interface configuration capabilities remain in place.

The `general-ui-config.xml` configuration file defines how the workflows will be visually structured on the different screens (groups and list of queues, etc.) and specifies the different objects and components that can be used in the different screens, such as custom panels for bulk screens and trade detail screens.

The `general-ui-config.xml` file has the following structure:

```
<queue-action-config>

<workflows>

<workflow ...>

<section ...>

<queue ...>

</section ...>

</workflow ...>
```

```
...
</workflows>
<search-queue>
...
</search-queue>
<quick-search-queue>
...
</quick-search-queue>
<bulk-panels>
<panel ... />
...
</bulk-panels>
<trade-panels>
<panel ... />
...
</trade-panels>
<audit-filters>
<filter ... />
...
</audit-filters>
</que-action-config>
```

Workflow View and Queue Lists

The purpose of the `<workflows>` section is to visually structure the different workflows into lists of queues and group them under different sections to organize the display. The configuration is rendered by a JSP screen, `queues.jsp`.

For each queue in a workflow, the next steps that a user can manually send a trade to can be defined. A PI variable can be specified to capture the selection for use by workflow transitions.

The `<workflows>` node can take any number of `<workflow>` child nodes. A `<workflow>` node has the following attributes.

Attribute	Required/ Optional	Description
name	Required	Workflow name, such as Outbound.

label	Optional	Workflow title to display.
tooltip	Optional	Quick description that displays when you hover over the option in the user interface.
nextStepVariable	Required	Name of the variable to be populated with the next step for routing purposes.

Example

```
<workflow name="Outbound" label="Outbound Queue List"
tooltip="Workflow tracks documents within Scrittura" nextStepVariable="queueRoute" >
```

The `<workflow>` node can have as many <section> child nodes as required. A `<section>` node has the following attributes.

Attribute	Required/ Optional	Description
name	Required	Queue name, which must match the name attribute of the corresponding `<queue>` element defined in `scrittura-config.xml`.
label	Optional	Section title to display.
tooltip	Optional	Quick description that displays when you hover over the option in the user interface.

Queues belonging to a section are defined as `<queue>` nodes and are children of the `<section>` node. A `<queue>` node has the following attributes.

Attribute	Required/ Optional	Description
name	Required	Workflow name, such as Outbound.
tooltip	Optional	Quick description that displays when you hover over the option in the user interface.

The following can be defined for each queue.

- **Action buttons** are available in trade detail screens, such as Save, Save & Forward, Close, and so on. Entering an annotation can be enforced for each of the defined buttons. This functionality works in conjunction with a specific action panel.
- **Next steps** provide the definition of the list of next steps that a user can send a trade to upon Save & Forward. Upon action, the PI variable defined as the nextStepVariable attribute of the workflow tag is populated with the selection when a trade that can be used by the workflow transitions is moved.
- **Custom sections** can be defined in case further custom configuration is required for the various custom panels.

Action buttons are defined as `<action-button>` tags under the `<action- buttons>` element. The `<action-buttons>` node has the following attributes.

Attribute	Required/ Optional	Description
annThread	Optional	Annotation thread to display in the trade detail screen with the possibility to enforce annotation input in that thread when leaving the Trade Detail screen.
annLabel	Optional	Option label that displays in the user interface. The name of the thread can be quoted using `{annThread}`.

`<action-button>` tags have an empty body and the following attributes.

Attribute	Required/ Optional	Description
id	Required	Action identifier.
display	Required	Button label.
endorceAnn	Optional	Boolean flag that forces users to enter an annotation when this button is actioned. Default: `false`
inRoles	Optional	Comma delimited list of roles for which this button will display. If unspecified, the button will display for all users except the ones defined by the `excludeRoles` attribute.
excludeRoles	Optional	Comma delimited list of roles for which this button will not display. If unspecified, the button will display for all roles defined by the `inRoles` attribute.

Next steps are defined as `<next-step>` tags under the `<next-steps>` element. `<next-step>` tags have an empty body and the following attributes.

Attribute	Required/ Optional	Description
value	Required	Value that is assigned to the next step variable.
display	Optional	Label that displays in the user interface for this action.
singleTradeHandler	Optional	Custom process handler run for each trade when this option is selected.
inRoles	Optional	Comma delimited list of roles for which this step will display.
excludeRoles	Optional	Comma delimited list of roles for which this step will not display.

Example

```
<workflows>
<workflow name="Outbound"
label="Outbound Queue List" tooltip: "Outbound Workflow Queue"
nextStepVariable="queueRoute">
<section name="ConfirmValication" label="Confirm Validation" tooltip="Confirm
validation.">
<queue name="Pending Review" tooltip="First review queue.">
<action-buttons annThread="Comments">
<action-button id="Save"
display="Save" inRoles="admins" excludeRoles="" />
<action-button id="SaveAndForward"
display="Save &amp;amp; Fwd" inRoles="admins" excludeRoles="" />
</action-buttons>
<next-steps>
<next-step value="valid1stLevel"
display="Validation 1st Level" inRoles="admins" excludeRoles='" />
<next-step value="vregen"
display="Re-generate"
</next-steps>
</queue>
</section>
<workflow>
</workflows>
```

Custom sections are defined as `<custom-section>` tags under the `<queue>` tag. A `<custom-section>` tag has the following unique attribute.

Attribute	Required/ Optional	Description
id	Required	Custom section identifier.

Custom sections may contain a list of parameters that can be used for custom purposes. Each parameter is defined as a `<param>` tag, child of the `<custom- section>` tag, `<param>` tags have empty bodies and the following attributes.

Attribute	Required/ Optional	Description

name	Required	Name of the parameter.
value	Optional	Value associated with the parameter.
display	Optional	Label that displays in the user interface for this action.
singleTradeHandler	Optional	Custom process handler run for each trade when this option is selected.
inRoles	Optional	Comma delimited list of roles for which this action will display. If unspecified, the action will display for all users except the ones defined by the **excludeRoles** attribute.
excludeRoles	Optional	Comma delimited list of roles for which this action will not display. If unspecified, the action will display for all roles defined by the `inRoles` attribute.

NOTE: Custom sections are a configuration framework in place for custom use in case custom panels require configurable parameters. No logic is actually coded in the core platform, but must be handled by the client code.

Example

```
<custom-section id="subpanel">
<param name="var1"
value-"var1" display="var1" processHandler="" inRoles='' excludeRoles="" />
<param name="var2"
value-"var2" display="var2" processHandler="" inRoles='' excludeRoles="" />
</custom-section>
```

Bulk and Trade Panels

It is possible to define and configure the panels to be used in bulk screens and trade detail screens. Panels for bulk screens are defined as `<panel>` tags under `<bulk-panels>`, whereas panels for trade detail screens are defined as `<panel>` tags under `<trade-panels>`.

A `<panel>` tag has the following attributes.

Attribute	Required/ Optional	Description
name	Required	Name of the panel, referenced in the views defined for queues in `scrittura- config.xml`.

page	Required	JSP that implements this panel.
bulkTradeHandler	Optional	Only available for bulk panels. This attribute associates a handler to the panel.

Example

<panel name="lnk"

```
page="/jsp/structures/panels/linkComponentPanel.jsp"
bulkTradeHandler="com.iwov.gcm.scrittura.web.queue.processhandlers.StructLink" />
```

Panels can be referenced in scrittura-config.xml for bulk screen and trade detail screen views. The base JSPs are bulkBase.jsp and tradeDetails.jsp is provided with the distribution The panels parameter of these pages is a comma delimited list of panel names (as defined in `general-ui-config.xml`) that defines which panels will display in the user interface.

```
<view name="Bulk Review"

type="bulk"

view="/jsp/bulkBase.jsp?panels=next"/>

<view name="Review"

type="custom"

view="/jsp/tradeDetails.jsp?panels=act,eco"/>
```

The following examples of pre-wired bulk panels are available (all are located under `/jsp/structures/panels`).

JSP	Description
editStructurePanel.jsp	Edit a structure.
addComponentPanel.jsp	Add a component to a structure.
linkComponentPanel.jsp	Link components as a structure.
unlinkComponentPanel.jsp	Unlink components from a structure.

The following examples of pre-wired trade detail screen panels are available (all are located under `/jsp/panels`).

JSP	Description
structurePanel.jsp	Structure information.
actionPanel.jsp	Action panel that displays action buttons.
economicPanel.jsp	Economic panels.
docEditPanel.jsp	Document editing panel.
docManager.jsp	DocManager documents.

miscPanel.jsp	Access to history and product dump.

Panel JSPs and Process Handlers

From a JSP perspective, panels are fragments of JSPs. Bulk panels are nested within a single general form (located in `/jsp/queue/queue.jsp`) and cannot directly contain forms, whereas trade detail panels are independent and may contain their own forms.

Process Handlers are similar to event handlers. They are Java classes that handle any logic required upon submitting a form from a trade detail panel or bulk panel.

There are two types of process handlers that serve two distinct purposes:

- **Bulk Trade Processing** is performed from the bulk screen when the same action applies to a list of trades selected by the user. The action logic is performed by the process handler from which the action originates. This type of Process Handler is called a Bulk Trade Handler and is specified by the `bulkTradeHandler` parameter of the bulk panel itself.

  Bulk Trade Handlers extend the class `BaseBulkTradeHandler` located in `com.iwov.gcm.scrittura.web.queue.processhandlers.`

  The process handler logic is called by the bulk event queue_action that first performs the validation (handler's `validateBulk()` method) prior to running the processing itself (handler's `executeBulkProcessing()` method).

  In case the validation is unsuccessful, no processing stops is performed. The executeBulkProcessing() method may return `STOP_PROCESSING` or `CONTINUE_PROCESSING`, both being integer values defined in BaseBulkTradeHandler. In the first case, the processing will stop after executing the handler; in the second case the processing will continue with forwarding trades to their next destination. Note that the relevant routes should be in place in the workflow. The `executeBulkProcessing()` method always executes within the event transaction, hence the whole transaction will be rolled back if the processing fails for any of the trades.

  Alternatively, you may want to isolate the processing for each trade to be performed within its own transaction. As a result, any failure during the trade processing will just impact the trade currently processed rather than the whole series. Such processing specific to a single trade is performed by the `executeTradeSpecificProcessing()` method.

  Depending on what action the bulk panel is supposed to perform, you should either override `executeTradeSpecificProcessing()` or `executeBulkProcessing()`, but not both on the same handler.

- **Single Trade Processing** applies when a trade is forwarded to its next destination, when clicking "Save & Forward" from the trade detail screen or, as part of a bulk move, from the queue screen. Prior to the actual move, some logic can be applied that is performed by the process handler associated to the selected next step. This type of Process Handlers is called a Single Trade Handler and is specified by the `singleTradeHandler` parameter to the next-step tags.

  Single Trade Handlers extend the class BaseSingleTradeHandler located in `com.iwov.gcm.scrittura.web.queue.processhandlers.` The handler logic is in the `executeSingleTradeProcessing()` method.

Search Results Pages

The queue-style search is the usual Scrittura search feature but whose results are displayed using the bulk screens, hence enriching the search with all bulk screen capabilities.

The queue-style search is a result page can be considered as a queue, which does not belong to any workflow, and can therefore be configured similarly to workflow queues.

As part of the configuration, the same sort of configuration as for workflow queues can be added to `general-ui-config.xml`. The Quick Search is configured under the `<quick-search-queue>` tag and the Advanced Search is configured under the `<search-queue>` tag—both are configurable in the same manner as the `<queue>` tag.

For information regarding full configuration of the Queue Style Search feature, see Search and Reporting, on page 220.

Audit Tracking Screens

Custom filters can be defined to aggregate audit annotations and display them in an organized manner.

A list of available filters is defined in `general-ui-config.xml`.

Pre-Wired Filters

All available filters are located in the package `com.ipicorp.scrittura.web.tracking.filters`. The following filters are provided with Scrittura.

Filter Class	Description
FilterAmendedFields	Displays the audit for amended values.
FilterDocumentGeneration	Displays the audit for document generation.
FilterQueueArrival	Displays queue arrival events.
FilterSignatures	Displays the signature-related audit.
FilterTradeEvents	Displays trade events.
FilterUserWorkflowMovements	Displays user workflow actions.

Extension Mechanism

Custom filters can be created and added to the list of available filters. A custom filter should extent the abstract class `AbstractFilter`, located in the package `com.ipicorp.scrittura.web.tracking.filters`.

The following methods must be implemented.

```
public abstract void setFilterCriteria();

public abstract TrackingEntry parseEntry(AuditData ad);
```

The `setFilterCriteria()` method lets you define filter criteria on the following `AuditData` fields when parsing audit records.

- actionType
- action
- username
- activity

Methods are available in `AbstractFilter` to set up the required criteria.

```
public void setFilterActionTypes(String [] values, String [] ops, boolean orClause);
```

```
public void setFilterActions(String [] values, String [] ops, boolean orClause);
```

```
public void setFilterUsernames(String [] values, String [] ops, boolean orClause);
```

```
public void setFilterActivities(String [] values, String [] ops, boolean orClause);
```

The corresponding SQL query is generated behind the scenes using the method information. For these methods, the following parameters are used.

Parameter	Description
values	Array of values for the corresponding criteria.
ops	List of corresponding operators, (such as IN, NOT IN, etc.) to build the SQL clauses. If null is passed instead, the operators will default to "=", or LIKE.
orClause	When set to true, coordinates the clauses with an OR.

The `parseEntry()` method takes an `AuditData` object as an input and must return a new `TrackingEntry` object. This lets you unify and reword the existing audit records.

The following `TrackingEntry` constructors are available.

```
public TrackingEntry(Date date, String type, String user, String action);
```

```
public TrackingEntry(Date date, String type, String user, String varname, String
oldValue, String newValue);
```

The following parameters are used.

Parameter	Description
date	Audit record date.
type	Audit record type.
user	User who triggered the audit.
action	Audit action.
varname oldValue, newValue	Variable name along with its old and new values.

Bulk Screen Configuration

The Bulk Screen framework is a user interface framework that lets users act on trades in bulk, performing business tasks involving multiple trades in single interactions. The business tasks that can benefit from this framework include supervisor signing, allocating, and dispatching multiple trades in one action.

Bulk Screen Functionality

The configuration of menus, queues, and columns displayed in Queue Views are an integral part of the Scrittura core platform. This configuration is established in the `scrittura-config.xml file`. The Bulk Screen framework introduces an enhanced level of flexibility to the Scrittura user interface by enabling complete control of the following aspects of queue screen appearance.

- **Style**. Select custom styles and preferences
- **Filtering**. Control dynamic column filtering
- **Display**. Access extensible formatting capabilities for column fields
- **Data Mapping**. Control type of data displayed

The configuration of the Bulk Screen framework is performed in a separate configuration file, the `scrittura-queue-view-config.xml` file, instead of the `scrittura-config.xml` file. The system combines the configuration set in both files to create the resulting queue screens.

Custom Action Panels

Custom action panels can be defined based on implementation requirements. A typical custom action panel can specify further attributes of an action button in order to apply the attribute and/or perform the action on the list of selected trades.

When you perform a custom action on items in bulk, a progress screen displays, informing you of the success or failure of the action on each trade. If an error occurs during the attempted action, that information displays on- screen.

Any number of custom action panels can be defined and included in one or more Bulk Screens. The development and integration of custom action panels are detailed in the sections within User Interface Configuration, on page 157.

Navigation Bar

The navigation bar identifies the following:

- Number of items displayed per page.
- Current page, with the ability to switch directly to a different page.
- Number of trades displayed per page.
- Current filter applied to the page, with the ability to switch filters.

- Create a new filter.
- Update a filter.
- Delete a filter.
- Export the filter as a Microsoft Excel file.

Settings are applied to an individual queue only. Different settings can be applied to different queues.

Trade List

The trade list occupies the main portion of the screen. A check box displays on each line, letting the user select the trade and apply bulk actions defined in the custom action panels.

The queue view header, located at the top of the trade list, contains the title of the queue as well as the headers of the columns that display in the trade list.

The drop-down box included with a column header identifies a filter for that column. Drop-down filters similar to those in Microsoft Excel can be configured on each individual column for rapid filtering and selection of the desired trade set. These filter settings must be configured for each individual queue. The check box on the left allows the user to select all trades displayed on the page. Settings defined by a user apply only to that user.

Using the icons in the queue view header, you can refresh the screen and select which columns to display in the queue. When the Columns Visibility icon is clicked, a pop-up dialog displays letting you select the columns to display in the trade list.

Queue Screen Configurations

The queue screens are compiled based on the configuration set in the following files located in the `/config Scrittura` directory.

- `scrittura-config.xml`
- `scrittura-queue-view-config.xml`

Scrittura combines the configurations set in both files to create the resulting queue screens. The base settings specified in the scrittura-config.xml file are enhanced with those specified in the `scrittura-queue-view- config.xml` file.

Configure scrittura-config.xml

The `scrittura-config.xml` file controls the configuration of queues, columns, and views that appear in the screen.

Queue Columns

The system relies on the `scrittura-config.xml` file to display the different menus and queues that comprise each menu. The following example shows a sample configuration showing only parts of menu, queue, and columns definition.

Example

```
<menu name=”Credit”>
```

```
<queue name=”Rejected” activity=”standardOutbound.Queue”>
<column display=”Cpty Name” variable=”counterparty” />
</queue>
</menu>
```

Queue View

The Bulk View used for a queue and the list of custom action panels to activate are specified under the `<queue>` element.

Example

```
<queue name="Validation" activity="standardOutbound.Validation">
<column ...>
...
<view name="Bulk Review" type="bulk"
view="/jsp/bulkBase.jsp?panels=next,tds"/>
</queue>
```

The base page is `bulkBase.jsp`, located in the /jsp directory of the Scrittura Web container. The panels parameter in the view URI uses a comma delimited list as a value. The items in that list are aliases given to the different panels to be displayed for that screen.

The list of available custom action panels is defined in `bulkBase.jsp` as a mapping of aliases with the corresponding panel JSP. Custom action panels can be added to this list if required. Corresponding pages should be developed as fragments of JSPs. They will be included in the main bulk screen page by the framework.

Configure scrittura-queue-view-config.xml

The `scrittura-queue-view-config.xml` file controls the appearance of the queue screens, including the following aspects:

- Custom styles and preferences.
- Sources for the data displayed in screen (variable sources).
- Column definitions - built-in renderer for text and custom renderers.
- Queue definitions.

General Settings

General settings are attributes placed within the root element of the file, `<scrittura-queue-view-config>`.

The `<scrittura-queue-view-config>` element has the following attributes.

Attribute	Required/ Optional	Description
custom-header	Optional	Context-relative URL of a custom JSP fragment to include in the bulks screen page header. This is usually used to change the styles used for the screen layout.
custom-search-header	Optional	Context-relative URL of a custom JSP fragment to include in the search result page header. This is usually used to change the styles used for the screen layout.
items-per-page-list	Optional	Comma delimited list of numbers to be used as options in the "items per page" drop-down list.

Example

The content of `/include/bulkHeader.jsp` is used as the header for the bulk screens. The items-per-page-list attribute lets uses choose between displaying 20, 50, or 100 items per page.

```
<scrittura-queue-view-config custom-header="/include/bulkHeader.jsp" items-per-page-list="20,50,100">
...
</scrittura-queue-view-config>
```

Variable Sources Definition

The variable source allows the definitions of local variables from built-in sources. The built-in sources include FA tables and other Scrittura tables.

Example

Sample configuration for adding a variable source to retrieve information from the Annotations table.

```
<var-source name="Annotations"
parent-source="pi table="scrittura_annotation">
<simple-column-mapper>
<mapping var="id" column="annid" type="integer"/>
<mapping var="piid" column="piid" type="integer"/>
<mapping var="user" column="username" type="string"/>
<mapping var="tmst" column="timestamp" type="timestamp"/>
<mapping var="thread" column="thread" type="string"/>
<mapping var="text" column="text" type="clob"/>
</simple-column-mapper>
<join-condition>Annotations.piid = pi.piid</join-condition>
</var-source>
```

Columns Definition

The columns definitions, `<column-def>`, define the columns that display in the user interface. The values in this configuration section override any values configured in the `scrittura-config.xml` file for matching IDs.

The `<column-def>` element has the following attributes.

Attribute	Required/ Optional	Description
id	Required	The column ID used to refer to the column definition from queue definitions.
default-title	Optional	Default column title. This can be overridden in the column element that uses the definition. If not specified, the column ID is used as the title.
align	Optional	Cell content alignment; default is **left**.

A Variable Renderer must be specified for each `<column-def>` element, as a child node of the latter. Built-in variable renderers are specified using the `<var-renderer>` tag, whereas custom renderers are specified using the `<custom-renderer>` tag.

Example

```
<column-def id="agreementType" default-title="Agreement Type">
<var-renderer var="fa:agreementType" quick-filters="true"/>
</column-def>
```

The following attributes apply when using the built-in variable renderer, which displays only text in the user interface.

Attribute	Required/ Optional	Description
var	Required	Full variable name of the variable source name and the variable name, separated with a colon.
format	Optional	Format of the string (applies only to date and number variables).
max-length	Optional	Maximum number of characters of the cell content.
no-content-text	Optional	Text to render in the cell if the variable value is empty.
no-content-style	Optional	CSS style class name to use for cells with no content.
link-url	Optional	If specified, the cell content becomes a link pointing to the specified

queue-style-search-link-url		URL. The URL may contain references to variables. These two attributes are respectively used for bulk screens and search result page.
link-style-class	Optional	CSS style call name for rendering the link.
link-target	Optional	Link target frame.
tool-tip	Optional	Cell tooltip text.
sortable	Optional	Specifies whether the column supports sorting.
quick-filter	Optional	Specifies whether the column has a quick-filter drop-down list.
make-content-HTML-safe	Optional	Specifies whether the displayed cell content should escape HTML characters.

In certain situations, a custom renderer is required to display all of the screen elements instead of only the text.

The custom renderer feature is required to implement the following Java interfaces.

- CellRenderer
- CellRendererFactor

A custom renderer is defined by the <custom-renderer> tag, whose unique attribute, factory-class, takes the renderer fully qualified Java class name as a value.

Example

```
<column-def id="views" default-title="Views" align="center">
<custom-renderer
factory-class="com.iwov.gcm.scrittura.web.queue
.render.ViewLinksCellRendererFactor">
<property name="varType">
<value>allVars</value>
</property>
</custom-renderer>
</column-def>
```

Queues Definition

All queues to display in the bulk screens require an entry in the queue definitions section using the <queue> element. A <queue> element has the following attributes.

Attribute	Required/ Optional	Description
id	Required	Defines the queue ID, formatted as menuName.queueDisplayName. The menu name and the queue display name are defined in the `scrittura-config.xml` file.
default-sort-by	Optional	ID of the column used to sort the queue table by default. If not specified, the queue is not sorted.
default-sort-order	Optional	Sort direction for the column specified by the default-sort-by attribute. Values of this attribute are asc for ascending and desc for descending order.
item-view-url	Optional	URL of the default item view accessed by double-clicking the row containing the item.

Example

```
<queue id="Credit.Credit Enrichment" default-sort-by="arrivalTime" item-view-url="">
<column id="ticketId"/>
<column id="wfStatus"/>
<column id="counterpartyName"/>
<column id="requestDate"/>
<column id="agreementType"/>
<column id="arrivalTime"/>
<column id="views"/>
</queue>
```

Similarly, a single optional entry should be defined for the Scrittura quick search and advanced search, in order to benefit from the queue-style search result page. This is detailed in Queue-Style Search, on page 224.

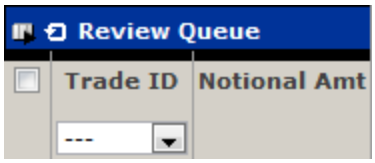
Each column to be configured for a certain queue is defined as a `<column>` element, child node of the corresponding `<queue>` element. The `<queue>` element has the following attributes.

Attribute	Required/ Optional	Description
id	Required	The column definition ID.
title	Optional	Column title displayed in the queue table header. If not specified, the default title from the column definition is used.
default-display	Optional	The default view of the column within the table. Values for this attribute are visible, hidden, and always- visible. If set to always-visible, the column cannot be hidden.

Columns displayed in the bulk screens can be specified in both the `scrittura-config.xml` file and the `scrittura-queue-view- config.xml` files. There is a subtle difference between these two possible configurations.

To fully benefit from the bulk screen functionalities (such as quick filters), columns should be defined in the `scrittura-queue-view-config.xml` file. Columns defined only in the scrittura-config.xml file cannot access the bulk screen features and will display by default on the right-hand portion of the row.

In the following illustration, the Trade ID column was defined in both the `scrittura-config.xml` and `scrittura-queue-view-config.xml` files and Notational Amt was defined in only the scrittura-config.xml file. The column filter is only available for the *Trade ID* column, and *Notational Amt* that displays by default on the right-side of the row.



Custom Cell Data Rendering Per Column

Custom renderer classes can be configured for each individual column, providing the ability to display color-coded or abbreviated text mapped out of actual underlying data, or to have tool tips or links leading to data. Such customizations are configured in the scrittura-queue-view-config.xml file and do not require any hard-coding or programming changes.

Example

The configuration of a column displaying the Boolean value of an FA variable `eConfirmSubmissionFailure`. The `CheckMarkCellRendererFactory` value displays a red check mark image when the value of the variable is true. If the variable is displayed on the Bulk Screen of a queue other than the eConfirm Submission Failure, clicking the check mark image results in the action view of the trade in the eConfirm Submission Failure queue.

```
<column-def id="eConfirmSubmissionFailure" align="center">

<custom-renderer factory-class=

"com.iwov.gcm.scrittura.web.queue

.impl.CheckMarkCellRendererFactory">

<property name="variableFullName">

<value>fa:eConfirmSubmissionFailure</value>

</property>

<property name="checkedLinkUrl">

<value>

/scrittura/controller?e=pi&amp;v=Review&amp;...

</value>

</property>
```

```
<property name="checkedToolTip">
<value>
Open in the eConfirm Submission Failure Queue
</value>
</property>
</custom-renderer>
</column-def>
```

Data Displayed

The fields to be displayed in the different columns of the bulk screens are defined in the `scrittura-queue-view-config.xml` file along with the actual trade data to which they map and any options attached to them (such as, optional or required).

The range of data that can be displayed in these Bulk Screens is not limited to Product Instance fields, as it is for the classic Scrittura Queue Views. Using the Bulk Screen framework, you can display the following data.

- Product Instance data
- Activity Item data
- FA Table data
- Custom fields derived from other data

Data displayed on the Bulk Screens can be accessed from the FA tables, but also from other tables as well, such as:

- WF_WORKITEM
- WF_ACTIVITYITEM
- SCRITTURA_PRODINST

Example

Configuration for a custom variable source to retrieve the latest annotation from a thread named Reviewer Comments.

```
<var-source
name="lastReviewerComment" parent-source="pi"
table="(SELECT piid, MAX(annid) annid, COUNT(annid) FROM scrittura_annotation
WHERE thread = 'Reviewer Comments' GROUP BY piid HAVING COUNT(annid) > 1)"
outer-join="true">
<simple-column-mapper/>
<join-condition> lastReviewerComment.piid = pi.piid
</join-condition>
```

```
</var-source>
```

Trade Detail Screen Configuration

Trade detail screens are displayed when a trade is opened, such as from a queue screen, as a result of the view event. The trade detail screen is a frameset, generally where the current document is displayed on the right side of the page with the actual trade details on the left side.

Distinct framesets and trade detail screens can be used for the different queues. This is configured in `scrittura-config.xml`, by the `<view>` tag, for each queue (defined by `<queue>` tags).

The configuration of the `<view>` tag is discussed in <menu> Node, on page 43.

The panels parameter of tradeDetails.jsp is the coma delimited list of the names of the panels to be displayed in the left pane. The names refer to the name attribute of the panels defined in the `<trade-panels>` section of `general-ui-config.xml`.

Example

```
In scrittura-config.xml

<view   name="Review"

type="custom" view="/jsp/tradeDetails.jsp?panels=act,str"
frameset="/jsp/viewFrameset.jsp"/>

In general-ui-config.xml

<trade-panels>

<panel name="str" page="/jsp/panels/structurePanel.jsp" />

<panel name="act" page="/jsp/panels/actionPanel.jsp" />

</trade-panels>
```

Two panels will display in the left pane of the trade detail screen, first the action panel as per `actionPanel.jsp`, then the structure panel as per structure panel.jsp.

The list of panels provided with the core Scrittura distribution is listed in Bulk and Trade Panels, on page 182. One particular panel, the Economic Panel, whose purpose is to display economic details of a trade, is explained in more detail in Economic Panels, below.

Economic Panels

The Economic Panel Framework provides fully configurable panels displaying economic data relevant to the current trade that can be inserted into the Trade Detail screens. Any number of panels can be defined, and criteria can be set to determine whether the panels should be displayed. The economic fields to be displayed can be configured for each panel.

The Economic Panel Framework is based on a configuration file, economic- `panels-config.xml`, located under the config repository in the Scrittura live folder.

The following is the structure of the configuration file.

```
<EconomicDetails>
```

```
<pane … >
<var …/>
…
</pane>
…
</ EconomicDetails>
```

Each <pane> element defines an economic panel to be displayed. There can be multiple <pane> elements inside the <EconomicDetails> element.

Economic fields to be displayed for each panel are defined using <var> tags under the <pane> element.

Panel Configuration for Economic Panel

A <pane> element has the following attributes.

Attribute	Required/ Optional	Description
name	Required	Panel display name
display-condition	Optional	Defines the Boolean condition that determines whether the panel is displayed. The condition is defined inside ${...}.

Example

Display a panel whose title is Counterparty Details, and the panel is displayed only if the variable productGroup has a value of CDS.

```
<pane name="CounterpartyDetails"
display-condition="${productGropup=='CDS'}>
</pane>
```

Field Configuration for Economic Panels

Inside a <pane> element, the <var> tag defines a field to be displayed in the panel. A <var> tag has the following attributes.

Attribute	Required/ Optional	Description
name	Required	Expression or variable to be displayed inside ${...}.
display-condition	Optional	Defines the Boolean condition that determines whether the panel is displayed. The condition is defined inside ${...}.

label	Optional	Defines the label to display for the field.
start-index	Optional	Start index to display arrayed variables.
end-index	Optional	End index to display arrayed variables.

Example

Display the value of the variable tradeOriginalID, preceded by the label

Trade Original ID. It is only displayed if the variable is null or empty.

```
<var name="${tradeOriginalID}"
display-condition="${tradeOriginalID!=''}" label="Trade Original ID"/>
```

The name attribute can be a field value, such as `${tradeOriginalID}`, or s simple expression. For more information on expressions, see Expressions, below.

Arrayed Variables

A Scrittura Product Instance can contain arrayed variables which can be displayed concisely in the economic panel framework.

To use arrayed variables, the generic wildcard "*" can be used. For an array called `myArray`, the name quoted in the economic panel framework will be `myArray[*]`. This applies to the name, the condition, and the label.

You must define the start and end indexes. The framework will perform the substitution with the actual index value while looping over the array of variables.

Although variable names must have brackets surrounding the asterisk `character ([*])`, brackets are not required in the label.

> **NOTE:** The following restrictions apply:
>
> - Only one-dimensional arrays with numerical indexes are supported.
> - Any asterisk character used in the label is replaced by the index value.
> - Any "[*]" string in the name or condition is replaced by the index value surrounded by brackets.

Example

```
<var name=${rate[*]}"
display-condition="${rate[*]!=''}" label="Rate #*"
start-index="1" end-index="4"/>
```

Expressions

Expressions can be used with the following attributes:

- condition (both var and pane elements)
- name (var attribute)

Expressions consist of expression operands, operators, and parentheses. Expressions must be nested within ${...} to be evaluated by the expression evaluation engine.

The following are valid expressions:

```
(var1 * (func1(var2, 2) + 25) < 100.5)
```

```
&& (var3 == "SOME TEXT")
```

```
var1 = array1[indexVar] + " CREATED: "
```

```
+ format_date(now(), "yyyy-MM-dd")
```

Expression operands can be literals, Scrittura variables, or function calls. Each operand has a particular type.

When the expression evaluation engine evaluates an operator with two operands, it brings both operands to a common type. The result of such sub- expression evaluation will also have a type, which will depend on the types of the participating operands.

Some operators require the operands to be of a certain type; for example, you cannot multiply a string unless the string can be parsed to a number.

The following literals are supported.

Literal Type	Examples
integer	1 536000
float	0.25 .5678 123.0
string	"some text" 'some text' ""
Boolean	true false

The following operators are supported, listed from the highest priority to the lowest.

Operator	Description
* /	Multiply and divide. Operands must be numbers or must be able to be converted to numbers. The result is a number.

+ - * /	Add and subtract numbers. The add operator can also be used with strings for string concatenation.
== != < <= > >=	Comparison operators. The result is Boolean.
&&	Logical "and". Operands must be Boolean. The result is Boolean.
||	Logical "or". Operands must be Boolean. The result is Boolean.
=	Variable assignment; this special operator tells the variable provider to set a variable value. The value to the left of the variable must be a variable name. The result is a new variable value.

NOTE: Unary operators, such as unary "minus" sign, and logical "not", are not supported.

The expression evaluation engine also provides some basic built-in functions, which include the following:

Function	Description
nvl(value, defaultValue)	If the specified value is null, an empty string, or a string consisting only of white-space characters, the function returns the provided default value. Otherwise, the value itself is returned.
string(value)	Converts the specified value to a string.
integer(value)	Tries to convert the specified value to an integer. If the value is a string, it will try to parse it. If it is a date, it will get the number of milliseconds since epoch. An empty string or a null value is converted to 0. The result is of Java type `Long`.
substring(string, start, end) substring(string, start)	Gets a substring of the specified string.
upper(string)	Transforms a string into uppercase.

lower(string)	Transforms a string into lowercase.
now()	Gets the current day's date. The result is of type Date.
date(string, format) date(millis)	Built-in functions that parse a date and return a result of type java.util.Date: The two-argument function parses a string applying the corresponding date format. The one-argument function creates a date from the specified number of milliseconds since epoch.
format(value,format)	Formats the specified value using the provided pattern and returns the formatted string representation. The value argument can be a date, in which case `SimpleDateFormat` is used, or a number, in which case `DecimalFormat` is used
abs(number)	Returns the absolute value of a number. The type of the result depends on the type of the argument.
if (condition,value1,value2)	If the condition is true, returns value1; otherwise, returns value2.
not(condition)	Negates the condition.

User Interface and API

A generic JSP panel, economicPanels.jsp, located in the Scrittura distribution under /jsp/panels, can be included in the trade detail screen.

Alternatively, you can develop a custom user interface that accesses the Economic Panel framework. Key classes are located in the package `com.iwov.gcm.web.beans` and include the following:

- `ViewTradeHelper`
- `EconomicDetailParam`

ViewTradeHelper

`ViewTradeHelper` is a helper class that provides methods to prepare the Economic Panel Framework for a specific Product Instance. The main methods are:

```
public Map prepareViewLeftTabsInfo( ProductInstanceLocal pi,

String[] includePanes, String[] excludePanes)

throws AuditRollbackException;

public Map prepareViewLeftTabsInfo(ProductInstanceLocal pi)

throws AuditRollbackException;
```

The required Product Instance is passed as the first parameter. Using the first method, you can specify which panels to include.

When called, these methods prepare the map of all panels and attributes according to the rules defined in the `economic-panel-config.xml` file.

They return a map of `ArrayList` objects with the following conditions:

- The map key is the panel name.
- The map values are `ArrayList objects`, representing the content of the economic panels. Each `ArrayList` is a list of `EconomicDetailParam` objects that represent the different fields to display.

EconomicDetailParam

EconomicDetailParam objects contain the information related to a specific field to display, including the following:

- Parameter name (meaning a variable)
- Parameter label
- Current value
- Previous value

The available methods are as follows:

Method	Description
public String getParamName()	Gets the parameter name.
public String getParamLabel()	Gets the parameter label.
public String getDisplayString()	Gets the current value.
public String getPreviousDisplayString()	Gets the previous value.

Chapter 7: BLogic Business Engine

This section details the use of BLogic, Scrittura's Business Logic Engine, which allows the configuration of business rules in the system.

This section contains the following topics:

BLogic Business Engine Overview

Scrittura's Business Logic Engine, or BLogic, is used to execute a business logic definition. A business logic definition is an ordered list of logic rules executed by the engine from top to bottom. Rules are ordered by priority over position when priority is specified.

In Scrittura, logic rules are typically executed against trades. Trade variables (such as, Product Instance or Message Ticket variables) are used as an input to evaluate a condition, which will either trigger actions or derive new variables necessary for the trade to continue its processing in the workflow.

Example

If a trade has counterparty "A" then it must be sent down the Scrittura workflow "B" and an email should be sent to employee "C" notifying them that the trade has been assigned to them. It could also set the variable "tradeManager" to "employee C", and assign address and contact information for counterparty "A" to other variables.

Given that rules files can contain as many rules as needed, there can be more than one BLogic workflow process and each process can run multiple rules files this allows for a phenomenal amount of business logic to be applied quickly and easily to a trade.

BLogic User Interface

BLogic rule files are saved as Excel spreadsheet (XLS format). Therefore, Microsoft Excel is used as a front-end user interface for BLogic, establishing a user-friendly environment for entering the logic rules. Rules can be organized in multiple Excel workbooks or spreadsheets.

Storage

Rules sheets are stored in DocManager in order to fully control their reload into the live system.

BLogic Integration with Scrittura

Using the BLogic Engine module from within a Scrittura workflow classtool is the most common integration of BLogic with Scrittura.

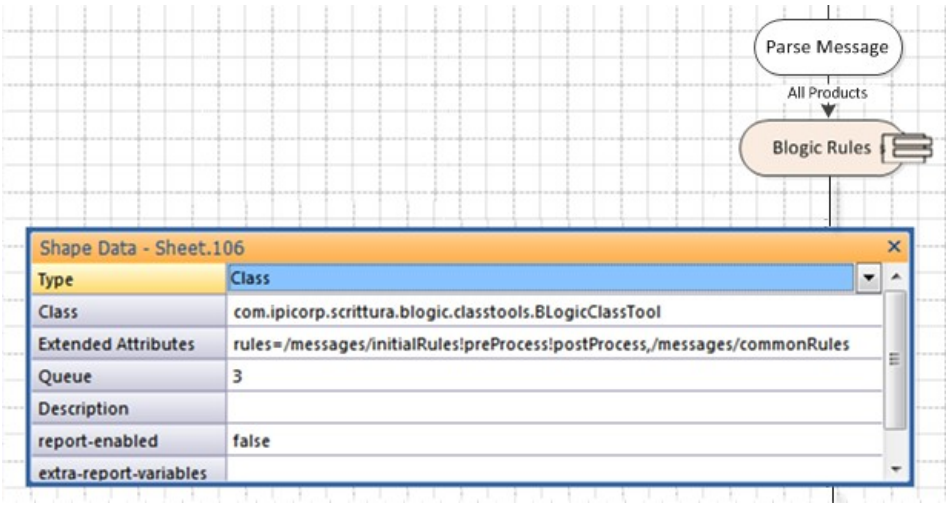
The following classtool is provided with the distribution.

Classtool	BLogicClassTool
Package	com.ipicorp.scrittura.blogic .classtools.BLogicClassTool
Attibutes	**rules.** The relative path of the rules file to be applied. The path is relative to the root BLogic directory. Alternatively this can be used to specify the Product Instance (PI) variable from which to read the location of the rules. The XML extension is not required for the file name.

Example of the BLogic Classtool in a Scrittura Workflow

The following illustration is a snapshot of the BLogic classtool within the Scrittura workflow. In this example, the class name is set to:
`com.ipicorp.scrittura.blogic.classtools.BLogicClassTool.`

The path to the rules file (located under the root BLogic directory) is specified as an Extended Attribute.



Example Specification of Multiple Workbooks in the Workflow System

The BLogic Workflow System allows the specification of multiple workbooks separated by a comma. For example:

```
/messages/initialRules,/messages/finalRules
```

It is also possible to specify specific/multiple sheets within a workbook by appending the name of the workbook with an exclamation mark, followed by the sheet name. For example:

`/messages/finalRules!finalSheet`

`/messages/finalRules!startSheet!finalSheet`

Multiple sheets from the same workbook (as specified in

`/messages/finalRules!`startSheet!finalSheet) load from both sheets, but the rules are run in order of priority.

To load sheets sequentially combine both commands. For example:

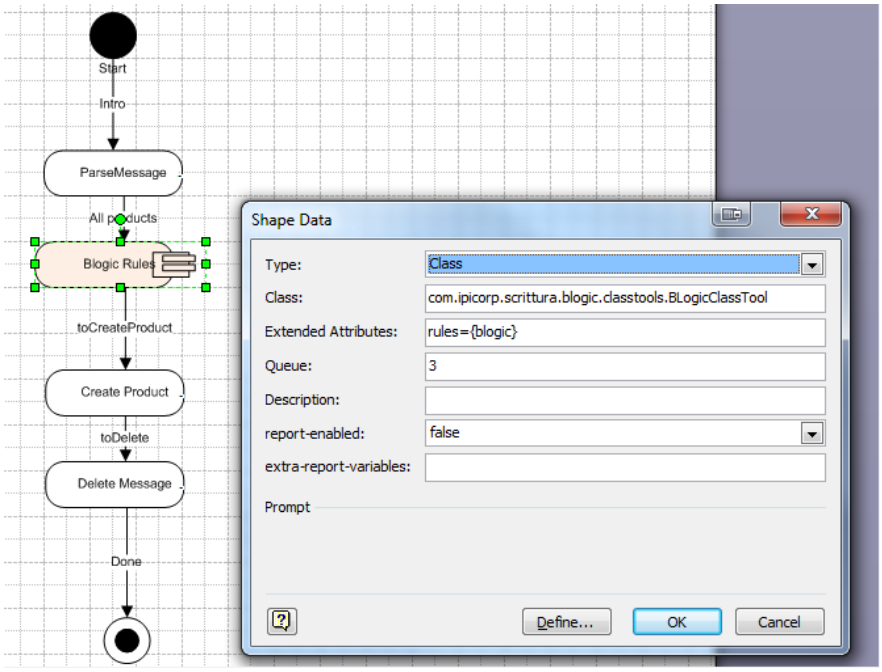
`/messages/finalRules!startSheet,/messages/finalRules!finalSheet`

For an example of the specification of multiple sheets and workbooks and the Rules path, see Extended Attributes in the illustration in Example of the BLogic Classtool in a Scrittura Workflow, on the previous page.

The ruleset may also be specified using a PI variable, as shown in the following illustration. This is done by setting the PI variable containing the ruleset path using curled braces. For example, rules={blogic} refers to the PI variable “blogic” that would contain the ruleset path such as

`/messages/finalRules!final. Example of Setting the PI Variable:`

Example of Setting the PI Variable:



BLogic General Configuration

The topics in this section describe the general configuration for BLogic.

BLogic Factory Initialization

To enable BLogic you must first set the appropriate options in `startup- config.xml`. The following is a sample configuration in `startup- config.xml`; the options are described following the sample.

```
<module name="blogic" enabled="true">
<folder>
<relative-path>/blogic</relative-path>
<docmgr-location>
Library Root/BLogic Rules
</docmgr-location>
<entity-type-path>BLogic Rules</entity-type-path>
<folder>
<relative-path>/messages</relative-path>
<docmgr-location>messages</docmgr-location>
<entity-type-path>BLogic Level 1</entity-type-path>
</folder>
</folder>
</module>
```

The following options should be set before running Scrittura.

- The attribute enabled of the module element must be set to true for BLogic to run.
- `relative-path` is the path on disk relative to the Scrittura root folder (either absolute-path or relative-path must be provided).
- `docmgr-location` is the full destination path to the folder in DocManger.
- `entity-type-path` must specify the entity type of the folder and all its parents at the desired DocManager destination; this parameter is required.
- `folder` is a recursive element with all the same sub-nodes as its parent. relative-path, docmgr-location, and entity-type-path are all relative to the parent folder.

> **NOTE:** For details about additional options available using startup-config.xml, see SetConfig Process Configuration, on page 373.

When you reload the Scrittura configuration through the `SetConfig` process, it prompts Scrittura to load the files contained inside the folder specified in `startup-config.xml` into DocManager.

When the BLogic system is first run, the specified rules and environment files are loaded from DocManager and parsed. The resulting ruleset is loaded into a cache that is refreshed whenever a new version of the rules file is loaded into DocManager.

Rules and environments may be loaded into DocManager through the standard DocManager user interface. They must be loaded into the folder created during the SetConfig process at the same path specified in the workflow.

> **NOTE:** When rules are loaded during a SetConfig, the files currently in DocManager with the same name/path are overwritten. Therefore, it is recommended that you only load the folder structure into the live folder and that the files be added manually from the user interface as required—after the folder structure has been created in DocManager during a SetConfig.

BLogic Environment

BLogic rules are organized in a folder hierarchy that allows the functional separation and ordering of different workbooks. This folder hierarchy would be typically created under the Scrittura live folder before being loaded in DocManager, as per the properties defined in `startup-config.xml`.

BLogic also requires its environment to be defined in specific files, `environment.xml`, located in the BLogic folder hierarchy. There can be one environment definition file at each level of the folder hierarchy, those being inherited from the parent levels. Placing an environment definition file under the BLogic directory in Scrittura live folder causes all other BLogic rule definitions in that directory to inherit the environment definition file.

> **TIP:** Each environment object may be declared only once in the environment hierarchy.

The environment is used to specify the items BLogic uses to evaluate and execute rules. For example, in Scrittura there is an Item Accessor class which is used to access the information in a trade. This needs to be specified in the environment file along with its name.

Example of environment.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE blogic-env SYSTEM

"file:///c:/opt/scrittura/dtd/blogic-env.dtd">

<blogic-env>

<object-type name="generic map"

accessor-class="com.ipicorp.scrittura.blogic

.impl.SimpleMapItemAccessor"/>

<condition-type name="is in queue"

factory-class="com.ipicorp.scrittura.blogic

.impl.SimpleComponentFactory">

<property name="objectClass">

<value type="class"> com.ipicorp.scrittura.blogic

.conditions.IsInQueueCondition

</value>

</property>
```

```
</condition-type>
<action-type
name="trigger workflow activity"
factory-class="com.ipicorp.scrittura.blogic
.impl.SimpleComponentFactory">
<property name="objectClass">
<value type="class"> com.ipicorp.scrittura.blogic
.actions.TriggerWorkflowActivityAction
</value>
</property>
</action-type>
</blogic-env>
```

The environment must specify all of the custom conditions, actions, and Item Accessors.

The `<object-type>` tag is used for declaring Item Accessors, an Item Accessor defining which object will be accessed by BLogic and how. The name attribute is what BLogic uses to retrieve the Item Accessor; the class attribute is simply the fully specified class name and reference which will be used to create instances of the Item Accessor.

The `<condition-type>` and `<action-type>` tags are used to declare custom conditions and actions. The name attribute is what BLogic uses to retrieve the condition or action; the factory class is the fully specified class name of the condition or action factory; the property child only accepts objectClass as a value and is for specifying the fully specified class name of the custom condition or action class.

Microsoft Excel Front-End

The following items present a global overview of the Microsoft Excel spreadsheet format and basic syntax, and rules to follow when entering BLogic rules.

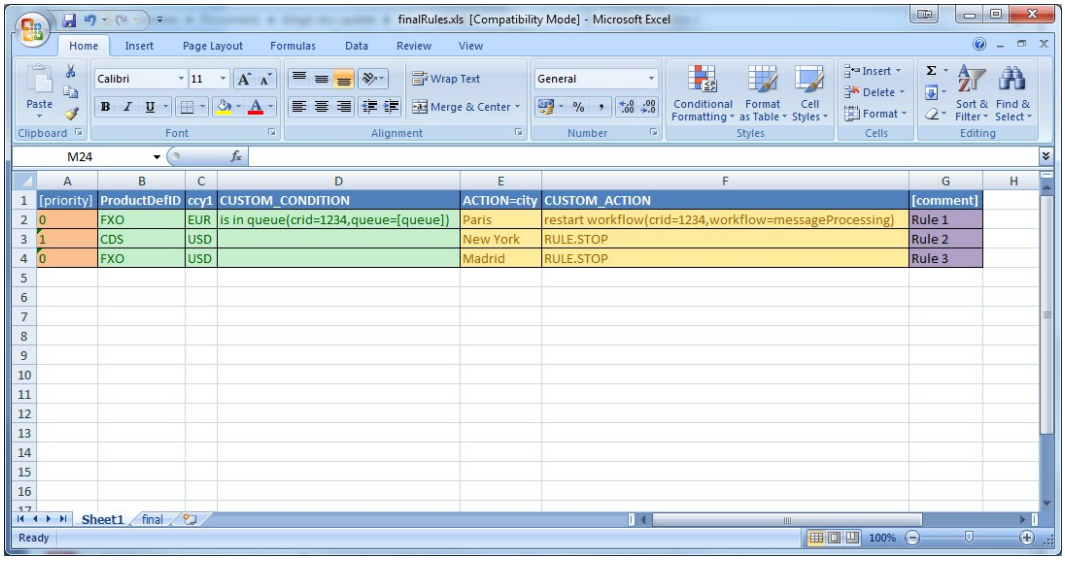
- Spreadsheets must be created using Microsoft Office Excel Professional 2003 or later.
- Documents must be saved with the extension .xls.
- Each spreadsheet must have at least one sheet containing rules.
- Each sheet should have a meaningful name.
- Only information related to business rules should be included in the spreadsheet.
- Any information in the spreadsheet is parsed; the parsing will therefore fail if information not relevant to the business rules is present.
- Each row in the spreadsheet is a separate business rule.
- Columns should indicate the variable name and the values of the variable for that rule.
- A whole blank row, column, or sheet are ignored and are not parsed.

- The first row must contain column headings, and all columns with information must have a column heading.
- The cells of the workbook must all be formatted as text.

Example Excel Business Rule Spreadsheet

The following is an example of an Excel Business Rule spreadsheet.

The first row indicates column headings. For columns that have variable names, it is important to note that the variable names are case-sensitive. If the variable name is in the wrong case, the rule does not execute correctly.



Rule Spreadsheet Columns

The following types of column headings are included in the rule spreadsheet.

- Priority Column, on the next page
- Variable Condition Columns, on page 211
- Custom condition columns display CUSTOM_CONDITION in the header. A custom condition column represents the custom conditions specified in the environment.xml file., on page 211
- Variable Action Columns, on page 212
- Custom Action Columns, on page 212
- Comment Column, on page 214

Example

The Rules in this example run in the following order; Rule1, Rule2, Rule3, Rule5, and then Rule4. The following pseudo code explains what each rule (in order top to bottom) is doing.

	A	B	C	D	E
1	[priority]	ccy1	CUSTOM_CONDITION	ACTION=city	CUSTOM_ACTION
2	0	EUR	isInQueue(CRID=1,queue =BLogic)	Paris	restart workflow(crid=C:TRD500,workflow=standardOutbound)
3	1	USD	isInQueue(CRID=42,queue =Generate)	New York	
4	1	USD		Madrid	
5	2		isInQueue(CRID=1,queue =End)	ATLANTA	
6	1	EUR			

1. IF `ccy = "EUR"` AND `isInQueue(CRID=1,queue=BLogic)`

   THEN set `city = "Paris"` AND restart workflow(`crid=C:TRD500`, `workflow=standardOutbound`)

   ELSE stop processing rules.

2. IF `ccy = "USD"` AND `isInQueue(CRID=42,queue=Generate)` THEN stop processing rules.

   ELSE set `city = "New York"` AND restart workflow(`crid=C:TRD007`, `workflow=generateMessage`).

3. IF `ccy = "USD"` THEN set `city = "Madrid"` AND stop processing rules.

4. IF `isInQueue(CRID=1,queue=End)`

   THEN set `city = "Atlanta"` AND stop processing rules.

5. IF `ccy = "EUR"` THEN stop processing rules.

> **NOTE:** Refer to this example when reviewing the sections about the individual spreadsheet columns.

Priority Column

The priority column, with a header of `[priority]`, orders the rules as specified, meaning that a rule of priority 0 is run before a rule of priority 1. Rules of the same priority are run in the order they are defined, from top to bottom, and in sheet order. The format for priority is `[priority]`.

The priority is carried over between sheets in the same workbook unless the sheet is specified separately within the workflow configuration. Following are two contrasting examples:

Example 1

The rules in the two sheets are combined, sorted, and evaluated according to their priority.

```
/messages/initialRules!sheet1!sheet2
```

Example 2

The rules in sheet1 are sorted first and evaluated, and then those in sheet2 are sorted and evaluated.

```
/messages/initialRules!sheet1,/messages/initialRules!sheet2
```

Variable Condition Columns

The variable condition column is set with the variable name in the header. This column sets a condition that compares any value entered with the one in the item being processed (this cell can use the operators explained in the operators table). In the example in Rule Spreadsheet Columns, on page 209, the variable condition column is titled ccy, which will compare the value in the cells to the value of the `ccy1` variable in the trade.

It is possible to use EL in these cells. To do so, surround the expression in curled braces, `'{'` and `'}'`. For example, to access a variable from the trade EL can be used, {partyCountry} would retrieve the value of the variable partyCountry from the trade.

Variable conditions are not evaluated for empty cells. There is a special EMPTY operand here which will only evaluate to true of the value is empty ("").

> **NOTE:** Column conditions are always linked by the AND operator.

Custom Condition Columns

Custom condition columns display `CUSTOM_CONDITION` in the header. A custom condition column represents the custom conditions specified in the `environment.xml file.`

For example, the "is in queue" condition refers to the custom condition defined in `environment.xml` under the `<condition-type>` node whose attribute is is in queue.

Custom conditions can be created as long as they implement the Condition interface (`com.ipicorp.scrittura.blogic.Condition`).

The parameters for the Custom Condition are entered with the syntax

```
(param1-name=param1-value,param2-name=param2-value)
```

where `param1-name` is the name of the parameter to be set and `param1- value` is the value that it should be set to (and similarly for the other parameters). The different parameters are comma separated.

To use PI variables as the parameters, the name of the variable needs to be entered between curled brackets: {variable}.

Example: IsInQueueCondition

`IsInQueueCondition` is a custom action that checks to see if the specified PI is in a specific queue.

`IsInQueueCondition` has the following parameters.

Parameter	Required/ Optional	Description
CRID	Required	The target PI CRID; can contain EL expressions.
queue	Required	The queue name pattern comprised of the workflow name (workflow process id) and activity name (activity id) separated with a dot. The activity name part can be "*", in which case any queue in the

		workflow is matched.
inverse	Optional	If set to true, the condition should check if the PI is not in the specified queue. Default is `false`.

Variable Action Columns

The format of the Action column heading is:

```
ACTION={Variable Name} OTHERWISE_ACTION={Variable Name}
```

These columns are used to set the variable specified by {Variable Name} to the value defined in the cell for that rule, and this variable must exist in the item being processed by BLogic. `ACTION` columns are executed if the rule condition is true, whereas `OTHERWISE_ACTION` columns are executed when the conditions evaluate to false

Any number of **Action** columns can be defined for each row, provided that they appear after the **Variable Name** column headings and update different variables.

Custom Action Columns

The format of the **Custom Action** column heading is:

```
CUSTOM_ACTION
```

```
OTHERWISE_CUSTOM_ACTION
```

Custom action columns represent the custom actions specified in the `environment.xml` file.

For example, the "restart workflow" custom action refers to the custom action defined in environment.xml under the `<action-type>` node whose name attribute is restart workflow. Custom actions can be created as long as they implement the Action interface (`com.ipicorp.scrittura.blogic.Action`).

`CUSTOM_ACTION` columns are executed when the rule condition is evaluated to true whereas `OTHERWISE_CUSTOM_ACTION` columns are executed when the rule condition is evaluated to false.

A native custom action is `RULE.STOP`. When defined and executed, this action prevents additional rules from being evaluated.

The parameters for `CUSTOM_ACTION` are entered with the syntax

```
(param1-name=param1-value,param2-name=param2-value)
```

where `param1-name` is the name of the parameter to be set and `param1- value` is the value that it should be set to (and similarly for the other parameters). The different parameters are comma separated.

To use PI variables as the parameters, the name of the variable needs to be entered between curled brackets.

Example: ApplyDataMappingAction

`ApplyDataMappingAction` is a custom action that finds a data mapping record given the key values and assigns the data mapping record values to the specified variables on the primary data object.

`ApplyDataMappingAction` has the following parameters.

Parameter	Required /Optional	Description
dataMappkingkeys	Required	Name of the data mapping.
valueVars	Required	Pipe-separated list if data mapping record lookup keys. Each element in the list consists of the key field name, "equals" sign (=), and the matching key value, which can contain EL expressions.
defaults	Optional	Pipe-separated list of values to assign to the primary data object variables in case no data mapping record satisfying the specified keys is found. Each element in the list consists of the variable name, "equals" sign (=), and the default value. Both variable names and the default values can contain EL expressions.

Example: `RestartWorkflowAction`

`RestartWorkflowAction` is a custom action that moves a PI back to the beginning of the workflow.

`RestartWorkflowAction` has the following parameters.

Parameter	Required/ Optional	Description
crid	Required	The target PI CRID; can contain EL expressions.
workflow	Required	The target workflow name (workflow process id).
startActivity	Optional	Name of the activity (activity id) where to start the PI. Default is Start.
ifNotInWorkflow	Required	Action behavior if no activity item is found for the PI in the workflow. Can be "ignore" to do nothing, "create" to create a new activity item, or "fail" to exit with an error, which is the default.

Example: `TriggerWorkflowAction`

TriggerWorkflowAction is a custom action that triggers the workflow to act on an activity item, sending it out of the current queue.

`TriggerWorkflowAction` has the following parameters.

Parameter	Required/ Optional	Description

crid	Required	The target PI CRID; can contain EL expressions.
workflow	Required	The target workflow name (workflow process id).

Comment Column

The comment column, with a header of `[comment]`, allows users to add comments or descriptions to each row under the column. This type of column is optional and its content is ignored by the rules engine.

Rule Condition Format

This section details the syntax that must be followed for condition cells. It explains the operators you can use as well as the format to follow when using them in rules.

The following is the general format of a cell for variable condition.

```
OPERATOR [VALUE ]
```

where

- `OPERATOR` is an operator.
- `VALUE` is the value to compare the variable with.

Valid Operators

The following operators can be used when defining a rule.

XLS Operator	Description	Example	Usable on
[value] value	Equals the operand value. Brackets are optional	[42], 42	Digits and characters
not[value]	Not equal to the operand value	not[USD]	Digits and characters
<[value]	The variable is less than the operand value	<[1000000]	Digits
<=[value]	The variable is less than or equal to the operand value	<=[512]	Digits
>[value]	The variable is greater than the operand value	>[1000001]	Digits
>=[value]	The variable is greater than or equal to the operand value	>=[513]	Digits
[value, value, value]	The variable is equal to one of those specified in the list of variables	[USD,EUR,GBP]	Digits and characters
not[value,	The variable is not equal to one of those	not[AUD,KZT]	Digits and

value, value]	specified in the list of values		characters
matches [value]	The variable matches the regular expression specified in the square brackets operand variable	[^\d*\w+\s?]	Digits and characters
not-matches [value]	The variable does not match the regular expression specified in the square brackets operand variable	not[^\w*\d*]	Digits and characters

Equals Operator

If the variable equals a certain value, enter that value in the cell.

In the following example, the **ProductDefID** and **ccy1** columns are **Condition** columns. The condition defined by the first row following the header is:

ProductDefID is **FXO** and **ccy1** is `EUR`.

The city variable is set to Paris if **ProductDefID** is **FXO** and **ccy1** is `EUR`.

	A	B	C
1	ProductDefID	ccy1	ACTION=city
2	FXO	EUR	Paris

Not Equals Operator

If a variable does not equal a certain value, enter that value within a set of square brackets preceded by the keyword not.

In the following example, `ProductDefID` not equal to FXO is shown using the value not `[FXO]`. This operator can be used for digits and characters.

	A	B	C
1	ProductDefID	ccy1	ACTION=city
2	not[FXO]	EUR	Paris

Less Than Operator

This operator can be used only for digits.

If a variable is less than a certain value, enter the value within a set of square brackets preceded by the less than symbol (<).

In the following example, **notional2** is less than 10.00.

	A	B	C
1	notional2	ccy1	ACTION=city
2	<[10.00]	EUR	Paris

Less Than or Equal To Operator

This operator can be used only for digits.

If a variable is less than or equal to a certain value, enter the value within a set of square brackets preceded by the less-than-or-equal-to symbol (<=).

In the following example, **notional2** is less than or equal to 10.00.

	A	B	C
1	notional2	ccy1	ACTION=city
2	<=[10.00]	EUR	Paris

Greater Than Operator

This operator can be used only for digits.

If a variable is greater than a certain value, enter the value within a set of square brackets preceded by the greater than symbol (>).

In the following example, **notional2** is greater than 10.00.

	A	B	C
1	notional2	ccy1	ACTION=city
2	>[10.00]	EUR	Paris

Greater Than or Equal To Operator

This operator can be used only for digits.

If a variable is greater than or equal to a certain value, enter the value within a set of square brackets preceded by the greater-than-or-equal-to symbol (>=).

In the following example, **notional2** is greater than or equal to 10.00.

	A	B	C
1	notional2	ccy1	ACTION=city
2	>=[10.00]	EUR	Paris

One of Operator

This operator can be used for digits and characters.

If a variable can be one of a list of values, enter the values within a set of square brackets with each value separated by a comma (,). Any spaces at the beginning or end of a value are removed automatically. No comma is required after the final value in the list.

For an empty value in the list, use the keyword `EMPTY`. For example, the list `EMPTY,FXO,CDS` means that the variable can be an empty cell, `FXO`, or `CDS`. In the following example, the variable can be `FXO` or `CDS`.

	A	B	C
1	ProductDefID	ccy1	ACTION=city
2	[FXO,CDS]	EUR	Paris

> **NOTE:** You can also use the format `FXO, ,CDS`, which allows the variable to be FXO, an empty cell, or CDS but using this syntax, the empty cell cannot be listed first.

Not One of Operator

This operator can be used for digits and characters.

If a variable cannot be one of a list of values, enter the values within a set of square brackets with each value separated by a comma (,). The opening square bracket is preceded by the word not. Any spaces at the beginning or end of a value are removed automatically. No comma is required after the final value in the list.

For an empty value in the list, use the keyword **EMPTY**. For example, **EMPTY,FXO,CDS** means that the variable cannot be an empty cell, **FXO,** or **CDS**. In the following example, the variable cannot be **FXO** or **CDS**.

	A	B	C
1	ProductDefID	ccy1	ACTION=city
2	not[FXO,CDS]	EUR	Paris

> **NOTE:** You can also use the format FXO, ,CDS which means that the variable cannot be FXO, an empty cell, or CDS but using this syntax, the empty cell cannot be listed first.

Matches Operator

This operator can be used for digits and characters.

If a variable matches the regular expression value, enter the value within a set of square brackets preceded by the word matches. In the following example, `ProductDefID` matches the string `FXO` or `CDS`.

	A	B	C
1	ProductDefID	ccy1	ACTION=city
2	matches[FXO,CDS]	EUR	Paris

Not Matches Operator

This operator can be used for digits and characters.

If a variable does not match the regular expression value, enter the value within a set of square brackets preceded by the phrase `not-matches`. In the following example, `ProductDefID` does not match the string `FXO` or `CDS`.

	A	B	C
1	ProductDefID	ccy1	ACTION=city
2	not-matches[FXO,CDS]	EUR	Paris

Cell Values

Cell values are standard strings, not nested within quotes, when a fixed value is used (such as 12.00, EQD, and so on).

PI variables can be specified by nesting their name within curled braces, such as {pivariable}.

When a cell is blank or contains only white spaces, the corresponding variable is ignored when evaluating the condition. Use the keyword **EMPTY** to test a variable against an empty value. The keyword **EMPTY** can be used for equals, not-equals, one-of, and not-one-of operators.

Example of a Rules Spreadsheet

The following shows the use of two rules in a Rules Spreadsheet.

	A	B	C	D	E	F
1	productGroup	ccy1	ACTION=city	OTHERWISE_ACTION=notional1	CUSTOM_ACTION	OTHERWISE_CUSTOM_ACTION
2	FXO		Paris	12.00	RULES.STOP	restartworkflow
3	EMPTY	USD	New York	1,066.95	restartworkflow	RULE.STOP

The first business rule is interpreted as follows:

```
IF (productGroup equals "FXO")

THEN set city to "Paris" AND stop rule engine

ELSE set notional1 to "12.00" AND run action "restart workflow"

Note that the ccy1 column for FXO is empty, so it is ignored.
```

The second rule is interpreted as follows:

```
IF (productGroup equals "") AND (ccy1 equals "USD")

THEN set city to "New York" AND run action "restart workflow" ELSE set notional1 to
"1,066.95" AND stop rule engine
```

Note that the **productGroup** column is used in the rule and is checked against an empty value.

For operators that allow only digits (less than, less than or equal to, greater than, or greater than or equal to), no spaces are allowed in the specified values. Only digits 0-9 and decimal separators, full stop (.) and comma (,) are allowed.

No characters or digits should be used after the closing square bracket (]). If any information is included outside the brackets, a validation error is thrown when Scrittura is loaded, and the rule spreadsheet will be rejected.

Rule Validation

When a rule file (.xls) is manually uploaded into a BLogic folder in DocManager, checks are run to ensure that the rules conform to specifications.

The checks include:

- Verifying that priorities are Integers.
- Ensuring that variables entered into the condition and action columns match their variable type, as defined in the Product Definition files.
- Validating custom conditions and actions, by verifying that they are defined in an environment.xml file, and ensuring that the specified classes are accessible.

- Making sure that any parameters provided with custom conditions and actions conform to the required syntax.

This is an automatic process and in the case of an error, a descriptive message is printed to the screen. The process is otherwise invisible.

The BLogic validator class provided is:

```
com.ipicorp.scrittura.blogic.BLogicValidator
```

For instructions on how to set up validation, see Custom Validator Classes, on page 137

Chapter 8: Search and Reporting

This section provides details on Scrittura's search and reporting capabilities, how to configure and customize them.

This section contains the following topics:

Scrittura Search Capabilities

Scrittura allows users to define and save their own searches (such as "show me all trades where the Counterparty name begins with 'X'"). These search definitions can be shared with other users or saved as 'personal' searches. There are different types of searches offering different features.

- **Advanced Search**. Provides selection across all trades by search on multiple variables and allows selection of variables to view.
- **Quick Search**. Works in a similar way to the Saved Search, but only allows selection on one variable at a time.
- **Queue Filter**. Provides search on only a single queue.

Results for Advanced and Quick searches can be displayed using the Scrittura classic search result view. Alternatively, and the recommended way, search results can be displayed using the Queue Style Search view, whose user interface is similar to that provided for viewing the contents of a queue, and provides the same enhanced capabilities to the search results.

Configure the Advanced and Quick Search

This section explains how to configure the Scrittura searches, make them available to users, or export them as reports.

Variables Available for Searches

The variables used as search criteria are defined in the scrittura- config.xml file as child <column> nodes of <search-columns>.

```
<search-columns>
<column variable="CommonReferenceID" display="Trade Reference" quicksearch="true"/>
<column variable="Party[B]" display="Counterparty" quicksearch="true"/>
<column variable="CounterpartyRefNo" display="Counterparty Ref" quicksearch="true"/>
<column variable="TradeDate" display="Trade Date" quicksearch="true"/>
```

```
<column variable="TradeType" display="Product Type" quicksearch="true"/>

<column variable="PartyFaxNo[B]" display="Counterparty Fax Number"
quicksearch="true"/>

</search-columns>
```

These variables appear in the order listed in the scrittura-config.xml file. These variables also appear in this order in the "Variable Name" and "Variable to Display" lists in the application Trade Search window.

The display attribute is used as column headers in the search result page.

> **NOTE:** The built in variable CommonReferenceID is required to be listed as a search column.

Scrittura supports a `<role>` attribute for each `<column>` so that a variable can be configured only to appear for certain users. This could be useful in cases where one set of users might be confused by seeing variables only relevant for another set of users. Also supported is the `<quicksearch>` attribute (true or false). This attribute determines whether or not a variable appears in the Quick Search drop down list at the bottom of the application window.

> **NOTE:** After making any changes to this list, the new scrittura-config.xml file must be reloaded using the Set Config screen.

Links to Views from Search Results

In their classic form, the Quick Search and Advanced Search result pages provide links to trade detail screens using any global views that have been defined in scrittura-config.xml with `<show-on-search>` set to true.

Using the Queue-Style Search, links to trade detail screens are defined in `scrittura-queue-view-config.xml`.

Saved Searches and Permissions

Scrittura has the ability for users to save and share their own searches. Saved search categories are pre-defined in `scrittura-config.xml` along with the associated permissions.

Export Searches

In their classic form, Quick Search and Advanced Search result pages can be exported to PDF, HTML, or CSV. For example the result of a search can be exported as a CSV file and then be used in Microsoft Excel for further formatting and sorting. To export the result of a search, select the appropriate `Result Type` and then click the **Search** button. The queue-style search offers export to the XLS format.

Scrittura can use the following reporting tools to perform the export:

- **Style Report**. The use of Style Report requires a license key.

- **Jasper Reports**. This is an open source Java software that integrates smoothly with Scrittura. Users define a Jasper Reports template, which is an XML file, using iReports. iReports is a Visual Designer that can be used to edit a Jasper Reports template.

To select which exporting tool to use, replace the following line in the `scrittura-config.xml` file:

```
<search-columns>
```

with the following for Style Report:

```
<search-columns exporting-tool="StyleReport">
```

or with the following for Jasper Reports:

```
<search-columns exporting-tool="JasperReport">
```

Using the Scrittura Search Functionality

Any variable declared in a Product Definition may be used in a search. For information about the configuration of the search variables, see Variables Available for Searches, on page 220.

Searches can be performed from the **Advanced Search** screen where users can define complex search criteria involving multiple variables or the Quick Search at the bottom of the screen, where users search against one variable at a time.

Define a Search

Search screens contain core elements for searching: Variable Name, Operator, Value, Variable to Display, and Sort. The 'Quick Search' frame does not include a 'Sort' option, since it is searching against a single variable.

- **Variable Name**. Select from a list of all variables configured for searching.
- **Operator.** Select the operator to be used to compare the selected **Variable Name** against the defined **Value**. Operator options include, but are not limited to, equals, less than, greater than, and LIKE.
- **Value.** Define the user-specified value to be compared to the selected **Variable Name.**
- Variable to Display. Select from a list of variables to be displayed in the resulting table, displaying a value for each "found" Product Instance.
- **Sort.** Select how the **Variable to Display** is to be used to sort the search output.

Comparisons and Data Types

For the purposes of searching, Text and Date variables are treated differently. Numerical data is essentially treated as text (unless it has been specified to be a date).

The following information describes how each operator works with each kind of variable.

Comparison Operator	Usage with Text variables	Usage with Date Variables

=	Finds trades for which the variable and the value specified are exactly the same.	Finds trades for which the variable date is the same as the value specified.
<	Finds trades for which the variable is "alphabetically less" than the value. (such as "a" is less than "b").	Finds trades for which the variable date is before the value specified.
>	Finds trades for which the variable is "alphabetically greater" than the value (such as "b" is greater than "a").	Finds trades for which the variable date is after the value specified.
<> (not equal to)	Finds trades for which the variable is not exactly the same as the value specified.	Finds trades for which the variable date is different than the value specified.
<=	The same as < except that it also matches for equality.	Finds trades for which the variable date is before or the same as the value specified.
>=	The same as > except that it also matches for equality.	Finds trades for which the variable date is after or the same as the value specified.
LIKE	Used to find values that are contained within variables. For example, "MyVar LIKE abc" would find any trades where the value of MyVar (perhaps "abcdef") includes the text abc within it.	Used to find values that are contained within the date when the date format is specified numerically as yyyymmdd. This operator expects values to be entered in pairs (yy,yy,mm,dd), so for the date 19981104, LIKE would match 9811 but would not match 811.

Dates should be entered as specified in the global or user preferences; that is, typically in the form 1 Feb 2013 (or however your system is configured); when using the LIKE comparison operator, the date should be specified in the format yyyymmdd (that is, 20130201).

In most cases it is easier and more intuitive to define specific ranges of dates, such as a combination of the conditions "TradeDate > 31 May 2013" and "TradeDate < 1 Jul 2013".

Case Sensitivity

It is possible to force Scrittura Quick Search and Advanced Search to be case- insensitive. For this purpose, set the use-upper-searches attribute of the

`<scrittura-config>` tag in `scrittura-config.xml`, as explained in Scrittura-config.xml File, on page 31.

Otherwise, case sensitivity in search comparisons depends on the configuration of the database server. For example, a SQL database can be specifically configured as "case sensitive" or "case insensitive."

In a case insensitive database, a query to return all instances where "Entity < b" would return instances where Entity begins with "A" or "a".

Wildcards

When using the LIKE operator, the '*' and '%' characters may be used as wildcards. They can be used in the Value field to represent any number of unknown characters. For example, searching for a*e will match "ae", "ade", and "abcde" ... but not "abcd".

When using LIKE for a search, wildcard characters are implicitly added to the beginning and end of whatever is entered in the Value field. For example, if you search for instances where a variable is "LIKE bcd", a value with the value "abcde" would match.

Built-in Date Variables

Scrittura includes the built-in dates today, yesterday and tomorrow. In addition, these dates can be 'added to' and 'subtracted from' to specify any date relative to today, yesterday or tomorrow (that is, "today + 14" would be used to indicate two weeks from today.)

For example, to find a variable that matches three days ago, one could enter the Value "today - 3", "yesterday - 2" or "tomorrow - 4".

No holiday or weekend calendars apply to these added and subtracted values.

Queue-Style Search

Scrittura's queue-style search reuses the bulk screen user interface for search result screens, bringing this rich user interface experience to the results of trade searches. It provides the following capabilities.

- Resizing of columns
- Dragging and dropping of columns
- Showing and hiding of columns
- Filtering the results by particular values of a column
- Performing bulk actions on search results

Queue-Style Search Features

In addition to the features already present in queue bulk screens, queue-style search incorporates several features which are useful for working with search results.

- **Archive and History**. In classic search results, the ability to archive any trade, or view a screen detailing its history, is available through a link posted next to each trade. With queue-style search, these same links are made available for every trade in the search results as well.
- **User Friendly Queue Names**. In bulk queue screens, every trade belongs to the same queue. In queue-style search, each trade in the set of results can belong to a separate queue, so each trade’s queue is displayed using the display name of each queue for user-friendly identification; this can be achieved by configuring the queue-style search queues.

- **Archived Trades and Trades-In-Progress**. Bulk queue screens display only trades that belong to a manual queue, but queue-style search results can contain both archived trades, and trades which are being processed by automatic processes. These trades are identified in the “Queue” column as “Archived (browse)” or “In progress…” respectively.

  The archived trades and trades-in-progress can be color-coded using CSS styles. For more information on using CSS style, see Configure the Styles of Trades, on page 229.

Activate the Queue-Style Search Functionality

`Queue-style` search and queue-style quick search are both activated by defining child elements of the <scrittura-config> element within scrittura-config.xml.

Example configuration

```
<scrittura-config>


...


<search-queue name=”Queue Style Search”
queue-style-only=false”>
<view name=”Bulk Review” type=”bulk”
view=”/jsp/bulkBase.jsp?panels=next,edstr,link”/>
</search-queue>
<quick-search-queue name="Queue Style Quick Search"
queue-style-only="false"
include-deleted-trades="true">
<column-set name="StandardColumns"/>
<view name="Bulk Review" type="bulk"
view="/jsp/bulkBase.jsp?panels=next,edstr,lnk"/>
</quick-search-queue>
---
</scrittura-config>
```

Queue-style search is activated by defining the <search-queue> or <quick- search-queue> elements are not specified, the corresponding search results will default to the classic search result view, respectively for advanced search and quick search.

Queue-style search is an optional feature. If the <search-queue> element is not specified, the classic search functionality is available in the advanced search screen, and if the <quick-search-queue> element is not specified, the classic quick search functionality is available on the quick search toolbar.

The `<search-queue>` and `<quick-search-queue>` elements have the following attribute.

Attribute	Required/ Optional	Description
name	Required	Defines the name of the queues. Can be used to differentiate from other queues already defined in the system. If not present, the default of "Queue Style Search" and "Queue Style Quick Search" are used for the search queue and quick-style search queues, respectively.
include-deleted-trades	Required	Required by default, the value is true. The search result will have deleted trades included.

For more information about the `<column-set>` element, see Define the Column Set forQueue-Style Search, on page 224; for more information about the `<view>` element, see Define the View for Queue-Style Search, on page 224.

Define the Column Set for Queue-Style Search

Using Advanced Search, you can select from a list of columns to display in the result set when they specify the parameters for the search. The set of columns from which they can choose is defined in the same manner as those available for the classic Advanced Search (defined in the `<search-columns>` element in `scrittura-config.xml`).

Using Quick Search, you define a set of columns that display every time a `queue-style` Quick Search is performed. This set is defined by the `<column- set>` child element (or the <column> child elements) of the `<quick- search-queue>` element as shown in the Example Activate the Queue-Style Search Functionality, on the previous page.

Define the View for Queue-Style Search

As with bulk screen queues, the view is defined for the Advanced Search and the Quick Search using a `<view>` child element of the `<search-queue>` and `<quick-search-queue>` elements, as shown in the Example in Activate the Queue-Style Search Functionality, on the previous page.

For more information about configuring the `<view>` element for bulk screen queues, seeBulk Screen Configuration, on page 187.

Define the Item View URL for Queue-Style Search

The URL to which the user is directed upon clicking on a trade can be customized under the <search-queue> and <quick-search-queue> elements in scrittura-queue-view-config.xml.

Example view URL configuration for queue-style search and queue- style quick searches

```
<scrittura-queue-view-config>

...

<search-queue

default-sort-by="arrivalTime" queue-style-search-item-view-url=
```

```
"/scrittura/controller?e=pi
&amp;v=Review
&amp;q={nonvar:derivedQ}
&amp;i={qm:workflowProcessId}.{qm:activityId}
.{qm:arrivalTransitionId}.{qm:workitemId}
&amp;currentPage={request:currentPage}">
<column id="arrivalTime"/>
</search-queue>
quick-search-queue
default-sort-by="arrivalTime" queue-style-search-item-view-url=
"/scrittura/controller?e=pi
&amp;v=Review
&amp;q={nonvar:derivedQ}
&amp;i={qm:workflowProcessId}.{qm:activityId}
.{qm:arrivalTransitionId}.{qm:workitemId}
&amp;currentPage={request:currentPage}">
<column id="arrivalTime"/>
<column id="visibleQueue"/>
</quick-search-queue>
---
</scrittura-queue-view-config>
```

The `<search-queue>` and `<quick-search-queue>` elements have the following attributes that define the click-through URL.

Attribute	Required/ Optional	Description
default-sort-by	Optional	Defines the name of the column by which the results of the search is sorted.
default-sort-order	Optional	Defines the default ordering, asc for ascendant ordering and desc for descendant. `Default: asc`
queue-style-search-item-view-url	Optional	Defines the URL to which the user is redirected when a trade is clicked in the queue-style search results. For more information about the format of placeholders for this URL, see Bulk Screen Configuration, on page 187.

Define a Custom Header for Queue-Style Search Pages

As with the custom-header attribute of <scrittura-queue-view- config> for bulk screen, a separate custom header can be specified for the queue-style search screens. This custom header is specified by the custom- search-header attribute of the <scrittura-queue-view-config> element within scrittura-queue-view-config.xml.

Configure User-Friendly Queue Names

Queue-style search include a custom renderer which can be used to translate the name of the queue in which a trade resides from the internal name used in the Scrittura database, to a user-friendly display name as used throughout the user interface.

This is similar to the custom renderers for columns in the bulk screens, which can be used to render the display of given columns in a manner particularly suited to the information in that column. For more information, see Columns Definition, on page 191.

To use this custom renderer for queue-style search, define a <column-def> for the CurrentManualQueue variable as follows:

- The custom renderer will be applied to the <column-def> which you use to display the current manual queue of a trade. In the following example, it is identified by the id attribute of CurrentManualQueue.
- The <custom-renderer> element for this <column-def> specifies a factory-class attribute of com.iwov.gcm.scrittura.web.queue.impl.QueueNameCellRende rerFactory, which creates the custom renderer.
- The following properties of this custom renderer must be defined:
  - Set processVariableFullName to fa:CurrentManualProcess, to store the name of the variable which holds the current process of the trade in the fast action tables.
  - Set queueVariableFullName to fa:CurrentManualQueue to store the current manual queue of the trade in the fast action tables.
  - Set quickFilterSupported to true, with a type attribute of boolean to indicate that it should be interpreted as a Boolean value.
  - Set tradeInProgressText to the text that you would like to display to indicate that a trade is in progress.
  - Set tradeArchivedText to the text that you would like to display to indicate that a trade is archived.

Example <column-def> configuration for user-friendly queue names

```
<column-def id="CurrentManualQueue" default-title="Queue">

<custom-renderer

factory-class="com.iwov.gcm.scrittura.web.queue

.impl.QueueNameCellRendererFactory">

<property name="processVariableFullName">
```

```
<value>fa:CurrentManualProcess</value>
</property>
<property name="queueVariableFullName">
<value>fa:CurrentManualQueue</value>
</property>
<property name="quickFilterSupported">
<value type="boolean">true</value>
</property>
<property name="tradeInProgressText">
<value>In progress...</value>
</property>
<property name="tradeArchivedText">
<value>Archived (browse)</value>
</property>
</custom-renderer>
</column-def>
```

Configure the Styles of Trades

Separate styles can be defined to assist in visually distinguishing the data in the results list. For example, you can apply an alternating background color to the rows to help distinguish rows from adjacent rows. Or, you can define styles for the display of trades to distinguish those which reside in manual queues, those which are in progress, and those which are archived.

To help visually distinguish a row from adjacent rows, two different background colors are used for the alternating rows of the table. When defining styles for trades in these rows, there are two different styles for each type of trade: one is identified with a name ending with the number 0, and the other is defined with a name ending with the number 1. You can define one style in these sets of two to have a different background color than the other to preserve the effect of distinguishing alternating rows by background color.

The pair of styles for default trades is defined in mainPage.css. If you want to customize these styles, the base version of mainPage.css is located under the stylesheets repository of `scrittura-web.war` within the `Scrittura` distribution.

To insert a customized version into your client code, place your custom mainPage.css under `custom-web\customJSPs\stylesheets\` within your client code.

The following style pairs can be configured for manual queues.

Style Pairs	Purpose

tr.table0	The default style for trades in a manual queue.
tr.table1	The default style for trades in a manual queue; this style can be used to distinguish alternating rows between `tr.table0` and `tr.table1`.
tr.table0 a:link	Style used when displaying links in rows for trades in a manual queue.
tr.table1 a:link	Style used when displaying links in rows for trades in a manual queue; this style can be used to distinguish alternating rows between `tr.table0 a:link` and `tr.table1 a:link`.
tr.table0 a:visited	Style used when displaying visited links in rows for trades in a manual queue.
tr.table1 a:visited	Style used when displaying visited links in rows for trades in a manual queue; this style can be used to distinguish alternating rows between `tr.table0 a:visited` and `tr.table1 a:visited`.
tr.table0 a:active	Style used when displaying the active link in rows for trades in a manual queue.
tr.table1 a:active	Style used when displaying the active link in rows for trades in a manual queue; this style can be used to distinguish alternating rows between `tr.table0 a:active` and `tr.table1 a:active`.
tr.table0 a:hover	Style used when links over which the user hovers the mouse cursor are displayed for trades in a manual queue.
tr.table1 a:hover	Style used when links over which the user hovers the mouse cursor are displayed for trades in a manual queue; this style can be used to distinguish alternating rows between `tr.table0 a:hover` and `tr.table1 a:hover`.

The styles for archived trades, and trades not in manual queues, are defined in queue.css. If you want to customize these styles, the base version of queue.css is located under the stylesheets repository of `scrittura- web.war` within the Scrittura distribution.

To insert a customized version into your client code, place your custom `queue.css` under `custom-web\customJSPs\stylesheets\` within your client code.

The following style pairs can be configured for archived trades.

Style Pairs	Purpose
tr.inactivetable0	Default style for trades not in a manual queue.
tr.inactivetable1	Default style for trades not in a manual queue; this style can be used to distinguish alternating rows between `tr.inactivetable0` and `tr.inactivetable1`.
tr.inactivetable0 a:link	Style used when displaying links in rows for trades not in a manual queue.

tr.inactivetable1 a:link	Style used when displaying links in rows for trades not in a manual queue; this style can be used to distinguish alternating rows between `tr.inactivetable0 a:link` and `tr.inactivetable1 a:link`.
tr.inactivetable0 a:visited	Style used when displaying visited links in rows for trades not in a manual queue.
tr.inactivetable1 a:visited	Style used when displaying visited links in rows for trades not in a manual queue; this style can be used to distinguish alternating rows between `tr.inactivetable0 a:visited` and `tr.inactivetable1 a:visited.`
tr.inactivetable0 a:active	Style used when displaying the active link in rows for trades not in a manual queue.
tr.inactivetable1 a:active	Style used when displaying the active link in rows for trades not in a manual queue; this style can be used to distinguish alternating rows between `tr.inactivetable0 a:active` and `tr.inactivetable1 a:active`.
tr.inactivetable0 a:hover	Style used when links over which the user hovers the mouse cursor are displayed for trades not in a manual queue.
tr.inactivetable1 a:hover	Style used when links over which the user hovers the mouse cursor are displayed for trades in a manual queue; this style can be used to distinguish alternating rows between `tr.inactivetable0 a:hover` and `tr.inactivetable1 a:hover.`

The following style pairs can be configured for trades not in manual queues.

Style Pairs	Purpose
tr.archivedtable0	Default style for trades not in a manual queue.
tr.archivedtable1	Default style for trades not in a manual queue; this style can be used to distinguish alternating rows between `tr.inactivetable0` and `tr.inactivetable1`.
tr.archivedtable0 a:link	Style used when displaying links in rows for trades not in a manual queue.
tr.archivedtable1 a:link	Style used when displaying links in rows for trades not in a manual queue; this style can be used to distinguish alternating rows between `tr.inactivetable0 a:link` and `tr.inactivetable1 a:link`.
tr.archivedtable0 a:visited	Style used when displaying visited links in rows for trades not in a manual queue.
tr.archivedtable1 a:visited	Style used when displaying visited links in rows for trades not in a manual queue; this style can be used to distinguish alternating rows between `tr.inactivetable0 a:visited` and `tr.inactivetable1 a:visited`.
tr.archivedtable0 a:active	Style used when displaying the active link in rows for trades not in a manual queue.

tr.archivedtable1 a:active	Style used when displaying the active link in rows for trades not in a manual queue; this style can be used to distinguish alternating rows between `tr.inactivetable0 a:active` and `tr.inactivetable1 a:active`.
tr.archivedtable0 a:hover	Style used when links over which the user hovers the mouse cursor are displayed for trades not in a manual queue.
tr.archivedtable1 a:hover	Style used when links over which the user hovers the mouse cursor are displayed for trades in a manual queue; this style can be used to distinguish alternating rows between `tr.inactivetable0 a:hover` and `tr.inactivetable1 a:hover`.

Queue Filters

Scrittura bulk screens offer the capability to apply filters to the results returned on screen. This also applies to the queue-style searches.

Global filters can be created, deleted, or amended from the bulk screens prior to being applied. It is also possible to add extra filtering at column level, provided this capability has been enabled for the column.

For more information about the configuration and capabilities of the bulk screens, see Bulk Screen Configuration, on page 187.

Jasper Reports and Style Reports

Scrittura end-users may define, save, and share their own trade search criteria for relatively simple queries (such as, show all the trades where A=B and Y=X). Most daily reporting can be accomplished with these simpler queries; the data can then be exported to Excel for further statistical grouping or analysis.

Scrittura has integrated a powerful open source java reporting tool Jasper Reports. Using its APIs you can develop custom reports, such as showing a pie chart representing the percentage of not yet issued trades by trade date.

Alternatively, it is possible to use the libraries of the commercial software Style Report Pro for development.

Custom report classes can be added to the client code of the Scrittura application. The custom report classes must be added to the `<reports>` element of `scrittura-config.xml`.

Scrittura has built in scheduler functionality. You can schedule to run any saved searches and any custom java reports individually or simultaneously. Custom java reports however have to implement the `com.ipicorp.scrittura.util.StandaloneReport interface`.

Example: Audit Activity Report using Style Reports

Audit information is stored in dedicated audit tables optimized for search or reports. This ensures better access to audited data.

The report class accesses audit information from the audit tables. In designing such a class, you must:

1. Set up the format of the report page.
2. Get the data into a 2-D array that the Style Report engine can understand.

When designing a Style Report class:

- Create two classes; one implements abstract clas `com.ipicorp.scrittura.web.reports.AbstractScritturaReport` and the other extends `AbstractTableLens`.
- `AbstractScritturaReport` expects one method, `runReport(String rType, boolean isHtml)`. This method sets up the title, fonts, headers, footers, and so on to format the report page.
- This class does is call a lens class that actually gets and formats the data to be displayed in the report.

  The lens class needs to create a 2-dimensional array 'table' and tell the Style Report engine how to parse each row and column in the table. Create the 2-D array in the constructor of the lens and tell it how to get to each cell of the 2-D table in the `getObject()` method. To do this, perform the JDBC query. For each row returned, parse the columns into a List and add the List to the 'row' List. This will create the 2-D table.
- Style Reports use the `getRowCount()` and `getColCount()` methods to determine how many rows and columns are in the table. Be sure to add any label or total columns in addition to the main table data.
- Style Reports call `getObject(row,col)` for each row and column, so it has to be told how to get a String for each (row,col) value in the 2-D array (in addition to column labels or totals).

BIRT Reports

Scrittura can also integrate with Apache BIRT for reporting purposes. Once reports are configured using the Scrittura BIRT module, they can be accessed by Scrittura and can be exported into DOC, XLS, PDF, or HTML format.

Prerequisites for Integrating BIRT

To integrate Scrittura with the BIRT connector module, you must download the following components.

- `BIRT runtime libraries (download from www.eclipse.org)`
- `jtds-1.2.5.jar file (download from jtds.sourceforge.net)`

Scrittura has been tested against BIRT 2.6.1, which is the version of BIRT described in this document.

Complete the following prerequisites for integrating with BIRT.

1. Application Server Startup, on the next page
2. Prepare Directories for BIRT Integration, on the next page

Application Server Startup

To integrate Scrittura with the BIRT connector module, you must add BIRT libraries to the Scrittura classpath.

The CLASSPATH section must contain the following entries.

- `coreapi.jar`
- `engineapi.jar`
- `modelapi.jar`
- `scriptapi.jar`
- `com.ibm.icu_4.2.1.v20100412.jar`
- `org.apache.commons.codec_1.3.0.v20100518-1140.jar`
- `odadesignapi.jar`

The `js.jar` file is also required. For the WebLogic application server, add the `js.jar` file as the first entry under the `PRE_CLASSPATH` section in the `setDomainEnv.cmd` file. For the JBoss application server, add it to the JBoss module XML file.

Prepare Directories for BIRT Integration

You must prepare directories for BIRT integration.

1. Copy the BIRT runtime libraries to the `\lib\ReportEngine` location in the Scrittura home directory. These libraries are provided with the BIRT release and are located under the \ReportEngine directory.
2. Copy the `jtds-1.2.5.jar` file into the `plugins\org.eclipse.birt.report.data.oda.jdbc_2.6.1.v201 00909\drivers` directory under the `\lib\ReportEngine` location.
3. Create the BIRT temporary directories as specified in the configuration. These directories will be used to generate HTML reports. For full details, see Configuration of BIRT Integration, below.

Configuration of BIRT Integration

The integration of BIRT reports into Scrittura is configured in the `birt- report-config.xml` configuration file, located in the `\config` directory of the Scrittura home repository.

The `birt-report-config.xml` configuration file must contain database connection information and a list of reports. The reports must be placed under `\reports` in the Scrittura home directory.

This file is composed of the following sections.

- `Database configuration (<database-config>)`
- `Directories (<directories>)`
- `Report list (<report-list>)`

Example BIRT integration configuration

```
<birt-report-config>
<database-config>
<data-source> org.eclipse.birt.report.data.oda.jdbc
</data-source>
<driver-class> net.sourceforge.jtds.jdbc.Driver
</driver-class>
<url>jdbc:jtds:sqlserver://localhost:1433</url>
<user>scrittura</user>
<password>c2NyaXR0dXJh</password>
</database-config>
<directories>
<report-dir>birt/birt-reports</report-dir>
<runtime-dir>birt/ReportEngine</runtime-dir>
<temp-dir>birt/temp</temp-dir>
<config-dir>birt/config</config-dir>
</directories>
<report-list>
<report name="report">
<title>Test Report</title>
<description>This is a test report</description>
<birt-design>test.rptdesign</birt-design>
</report>
</report-list>
</birt-report-config>
```

The `<database-config>` node is mandatory, has no attributes, and has the following child nodes. All child nodes have no attributes, their value being specified in their body.

Child Node	Required/ Optional	Description
data-source	Required	Data Source type, such as: `org.eclipse.birt.report.data.oda.jdbc`
driver-class	Required	Driver class, such as `net.sourceforge.jtds.jdbc.Driver`
url	Required	Database URL, such as

		`jdbc:jtds:sqlserver://localhost:1433`
user	Required	Login name for the database connection.
password	Required	Password for the database connection. The password can be AES-encrypted or remain unencrypted in the configuration file.

NOTE: If you want to deploy the same report to different clients, remove <data-source> tag from the report.

The `<directories>` node is optional, has no attributes, and has the following child nodes.

Child Node	Required/ Optional	Description
runtime-dir	Optional	BIRT runtime directory, located under the Scrittura home directory. Default: `birt`
report-dir	Optional	Directory under the BIRT runtime directory where rptdesign reports are copied. Default: `reports`
temp-dir	Optional	Temporary BIRT directory, located under the BIRT runtime directory. Default: `temp-birt`
config-dir	Optional	BIRT internal configuration folder. This folder will be located inside the BIRT runtime library folder if unspecified.

The `<report-list>` node is mandatory has the following attribute.

Attribute	Required/ Optional	Description
log-level	Required	BIRT Engine log level. Possible values: `SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST`, and `OFF`

The `<report-list>` node supports one or more <report> child nodes, each defining a report.

A `<report>` node has a single attribute, name, that must be unique. If it is not unique, only the last instance will be kept. The <report> node has the following child nodes. All child nodes have no attributes, their values being defined by their body.

Child Node	Required/ Optional	Description
title	Required	Title describing the report, to be displayed in the user interface.

description	Required	Short description of the report, to be displayed in the user interface.
birt-design	Required	BIRT report design file (with .rptdesign extension). The corresponding file must exist and must be located in the BIRT report directory defined by the report-dir attribute of the <report-list> node.

BIRT Password Encryption

The password defined in the BIRT configuration file can be AES-encrypted if required.

All password encryptions in Scrittura are done in the `startup-config.xml` configuration file. For more information, see Password Encryption, on page 379.

Scrittura User Interface Configuration for BIRT Integration

The Reports menu item displays a list of reports with titles, descriptions, and format options, including HTML, PDF, Word (DOC), and Excel (XLS).

The corresponding JSP, /jsp/viewReports.jsp, is launched by Scrittura controller upon reception of a URL with the following parameters.

Parameter	Value
event	birt
action	list or empty value

Selecting one of the available format options generates the report in the corresponding format. This is done using an HTTP call with the following parameters to Scrittura controller.

Parameter	Value
event	birt
action	run
format	html, pdf, doc, or xls
report	Name of the `rptdesign` report file name to process.

BIRT Report Design

Eclipse BIRT Designer is the recommended tool to create and configure reports in a user friendly manner. BIRT report design files use the `.rptdesign` extension.

The data model (including column name, display, data source, and so on) as well as the style and layout are defined in the `.rptdesign` files.

For details on designing BIRT reports, see the BIRT documentation.

Chapter 9: Static Data Framework

The Static Data Framework provided by Scrittura allows the definition and configuration of a data model along with the corresponding user interface in order to store and manage counterparty static data directly in Scrittura.

This section contains the following topics:

Static Data Framework Overview

The configuration and setup of the Static Data Framework requires the following steps:

1. Define the data model to implement.
2. Configure the framework to map this data onto in-memory objects that can be accessed from within the framework and define user permissions.
3. Create the database tables.
4. Configure the user interface to allow access by end-users from within the Scrittura application.

Define the Data Model

The Static Data Framework supports common relational data models. Any number of tables can be defined and linked together using one-to-many or many-to-many relationships. Uniqueness constraints can be added and fields can be made required or not.

To illustrate the Static Data Framework in use, a simplified data model is used throughout this chapter. In this model, a list of counterparty contacts is created and classified by products in order to hold the following data:

- General counterparty details including name, short code, address, and so on.
- Each counterparty trades certain types of products such as FX Options, Equities, and so on.

- For a specific counterparty, a list of contacts is associated with each product traded by the counterparty. Contact details include first name, last name, email address, and phone number.

In the following table, the header of each column is the name of the table, with its fields listed below, primary keys appear in bold, foreign keys in italic, and required fields are followed by a star.

SD_BASEVIEW	SD_COUNTERPARTY	SD_CONTACT	SD_PRODUCT
IDVIEW	**IDCPTY**	**IDCONTACT**	**IDPRODUCT**
IDCPTY *	LONGNAME *	FIRSTNAME *	CODE *
IDCONTACT	SHORTCODE *	LONGNAME *	NAME *
IDPRODUCT	ADDRESS	EMAIL	DESCRIPTION
	COUNTRY	PHONE	

In this example, the primary keys of the four tables are `IDVIEW, IDCPTY, IDCONTACT,` and `IDPRODUCT`. All other fields of the SD_BASEVIEW table are foreign keys to the other three tables.

After you configure the framework, the Static Data Framework will generate DDLs (Data Definition Language) in order to create the tables.

Data Mapping Configuration

The data mapping configuration file, datamapping-config.xml, is located in the /config directory of the Scrittura live folder and is structured as follows:

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE data-mappings PUBLIC "-//Scrittura//DTD Data Mappings XML V2.0//EN"
"@scrittura.home@/dtd/datamapping-config.dtd">

<data-mappings>

<value-type-definition … >

…

</value-type-definition>

<data-mapping … >

…

</data-mapping>

<data-mapping … >

…

</data-mapping>

…

</data-mappings>
```

The different data mappings are configured under the <data-mappings> root node, each child <data-mapping> node corresponding to a table in the data model. Using the data model example in Define the Data Model, on page 238, four data mappings are defined:

- BaseView
- Counterparty
- Product
- Contact

Any number of `<value-type-definition>` or `<data-mapping>` tags are permitted, but at least one `<data-mapping>` tag is required. Custom value types can also be defined in the `<value-type-definition>` tag.

Root Node

`<data-mappings>` is the root node of `datamapping-config.xml` and defines the various data mappings.

Additional columns will be added to the database tables in order to store additional record properties (such as the user who created the record, date it was created, and so on). Using `<data-mappings>`, that data can be mapped to existing fields of the data model if required.

The `<data-mappings>` node has the following attributes.

Attribute	Required/ Optional	Description
default-id-column	Optional	Default name for data mapping table column that contains the record id used as primary key. This field defaults to `REC_ID` and the corresponding column must be of type `LONG`.
priority-column	Optional	Name for the data mapping table column that contains the record priority. Defaults to `REC_PRIORITY`.
created-by-column	Optional	Name of the data mapping table column that contains login name of the user who created the record. Defaults to `REC_CREATED_BY`.
created-on-column	Optional	Name of the data mappings table column that contains the timestamp for when the record was created. Defaults to REC_CREATED_ON.
modified-by-column	Optional	Name of the data mapping table column that contains the login name of the user who last modified the record. Defaults to `REC_LAST_MODIFIED_BY`.
modified-on-column	Optional	Name of the data mapping table column that contains the timestamp for when the record was last modified. Defaults to `REC_LAST_MODIFIED_ON`.

The <data-mappings> node accepts <value-type-definition> and <data-mapping> nodes as child nodes.

Custom Value Types

Custom value types are specified under the <value-type-definition> node. Custom value types can be of two types:

- **Simple value type**

  ```
  <value-type-definition>
  <simple-value-type>
  <valid-values>
  …
  </valid-values>
  </simple-value-type>
  </value-type-definition>
  ```

- **Data mapping value type**

  ```
  <value-type-definition>
  <data-mapping-value-type>
  ...
  </ data-mapping-value-type>
  </value-type-definition>
  ```

  A Simple Value Type provides the ability to define a set of authorized values whereas the Data Mapping Type refers to one of the configured data mapping. The Data Mapping Value Type is used to define foreign keys.

  For <simple-value-type>, the list of possible values are specified by adding <valid-values> tags where the value to be matched is set in the value attribute. For example:

  ```
  <value-type-definition name="ProductFamilyType">
  <simple-value-type type="String" max-size="20">
  <valid-values>
  <valid-value value="Commodities"/>
  <valid-value value="Credits"/>
  <valid-value value="Equities"/>
  <valid-value value="FX Options"/>
  <valid-value value="Rates"/>
  </valid-values>
  ```

```
</simple-value-type>
</value-type-definition>
```

Data-Mapping Attributes

The `<data-mapping>` node includes the following attributes.

Attribute	Required/ Optional	Description
name	Required	Internal name used to identify this data mapping. Names must be unique throughout the configuration.
display-name	Optional	Data mapping name to be displayed in the user interface. If not specified, the internal name is used.
description	Optional	Data mapping description to be displayed in the user interface.
table	Required	The database table used to store the mapping records.

Example

```
<data-mapping name="Counterparty" display-name="Counterparty"
description="List of counterparties" table="SD_COUNTERPARTY"
use-cache="false">
```

Data Mapping Child Nodes

The <data-mapping> node defines the different tables of the data model that will be created. One tag corresponds to one table. The tag is configured by a series of attributes and the following child nodes.

- <access>
- <key-field>
- <value-field>
- <unique-group>
- <required-group>

User Permissions

Three roles should be defined in the application server in order to handle different levels of permissions in the Static Data Framework:

- DMoperators
- DMmanagers
- DMadmins

Specific user permissions are defined within the <access> node. Permissions are set against the different roles using <read> and <write> tags, for read and write permissions respectively. These tags take a single attribute, permission, which contains the name of the role.

Example

```
<access>
<read permission="DMadmins" />
<write permission="DMadmins" />
<read permission="DMmanagers" />
<write permission="DMmanagers" />
<read permission="DMoperators" />
</access>
```

> **NOTE:** Write permissions do not automatically grant read permissions; read permissions must be specified explicitly.

Key and Value Fields

Key and Value Fields are entries within the data mapping that map onto a field of the corresponding database table as specified in the <data-mapping> attributes. Key fields can be used in a database lookup request, value fields cannot.

Key Fields

At least one <key-field> is required within each data mapping. The following is a typical structure for a key field.

```
<key-field … >
<value-type>
<simple-value-type … >
<valid-values>
…
</valid-values>
</simple-value-type>
</value-type>
</key-field>
```

<key-field> has the following attributes.

Attribute	Required/ Optional	Description

name	Required	Field name for the framework API.
display-name	Optional	Field name for the GUI. If not specified, the API name is used.
description	Optional	Field description.
column	Optional	Database table column that contains the key value.
optional	Optional	Boolean value. By default, all fields are mandatory, and a value must be specified for them whenever a Data Mapping record is created. Setting this attribute to true removes this restriction on the field.
matchRecordId	Optional	Boolean value. In order to refer to a foreign field in a table linked by a foreign key, the value must be set to true.

The field type can be specified within a `<simple-value-type>` node, as a child node of `<value-type>`, itself within the `<key-field>` node. The `<simple-value-type>` tag can specify a list of authorized values similar to the `<value-type-definition>` tag.

The `<simple-value-type>` tag and has the following attributes.

Attribute	Required/ Optional	Description
type	Required	One of the following types must be specified: `String, Boolean, Integer, Long, Decimal, Date, or DateTime.`
max-size	Optional	For string types, this field indicates the maximum length of the value string; for decimals, the maximum length of the integer and the precision, separated by a comma.
min-size	Optional	For string types, this field indicates the minimum length of the value string.

Example

```
<key-field name="Name"
display-name="Counterparty Name" description="Counterparty Name"
column="COUNTERPARTY_NAME">
<value-type>
<simple-value-type type="String" max-size="20"/>
</value-type>
</key-field>
```

Value Fields

Value Fields can be added to a data mapping, but cannot be used as search criteria in a lookup request.

The `<key-value>` node has the following attributes.

Attribute	Required/ Optional	Description
name	Required	Field name for the framework API.
display-name	Optional	Field name for the GUI. If not specified, the API name is used.
description	Optional	Field description.
column	Required	Database table column that contains the field value.
optional	Optional	Boolean value. By default, all fields are mandatory, and a value must be specified for them whenever a Data Mapping record is created. Setting this attribute to true removes this restriction on the field.
value-constraint	Optional	Name of foreign key or "check" constraint used to limit field's valid values. If not specified and the framework needs a constraint because of the field type, default name generated by the framework is used. The default name is formed by appending "_FKn" or "_CKn" for "foreign key" or "check" to the table name.

Example

```
<value-field name="Description"

display-name=" Description" description=" Product Description "
column="DESCRIPTION">

<value-type>

<simple-value-type type="String" />

</value-type>

</value-field>
```

Unique Groups

The `<unique-groups>` tag allows combining keys into unique groups (sets a unique value constraint). Multiple key fields can be included in the same group, in which case a unique constraint is created for the combination of the participating key values.

<unique-group> has the following attributes.

Attribute	Required/ Optional	Description
name	Required	Group name
display-	Optional	Name for display purposes; if not specified the API name is used to

name		present the group to users.
constraint	Optional	Name of the corresponding database unique constraint. If omitted, table name appended with _UQn is used, n being an integer

<unique-group> is further defined by the required <field-ref> tag and its required field attribute. The field attribute must contain the name of a field to be included in the constraint.

Example

```
<unique-group name="IdCounterparty_PrimaryKey"
constraint="IdCounterparty_PrimaryKey">
<field-ref field="IdCounterparty" />
</unique-group>
```

Required Groups

The <required-group> tag specifies that one field is mandatory within a group of non-mandatory fields. This is achieved by grouping fields within a Required Group. As many Required Groups as needed can be defined for a Data Mapping.

<required-group> has the following attributes.

Attribute	Required/ Optional	Description
name	Required	Group name
display-name	Optional	Name for display purposes; if not specified the API name is used to present the group to users.
constraint	Optional	Name of the corresponding database unique constraint. If omitted, table name appended with _RG is used.

<required-group> is further defined by the required <field-ref> tag and its required field attribute. The field attribute must contain the name of a field to be included in the constraint.

Example

In this example, either CounterpartyCountry or CounterpartyCity must be specified. Attempting to insert a record with neither field specified will fail.

The Required Groups constraint is implied in addition to other constraints. If CounterpartyCountry is also defined as mandatory, it must be populated whether CounterpartyCity is populated or not.

```
<required-group name=”location_RG” constraint=”location_RG”>
<field-ref field=”CounterpartyCountry” />
<field-ref field=”CounterpartyCity“ />
</required-group>
```

Table Relationships

This section describes how to setup table relationships within the Static Data Framework. To establish a relationship between tables, a field of the parent table is used as a Foreign Key that refers to the Primary Key of the child table.

For example, a table might contain three foreign keys referring to three other tables.

Framework Configuration

A Foreign Key is specified using a `<key-field>` tag within a data mapping. This Key Field is defined as a `<data-mapping-value-type>` instead of a

```
<simple-value-type>:

<key-field … >

<value-type>

<data-mapping-value-type />

</value-type>

</key-field>
```

When used as a foreign key, `<key-field>` has the following attributes.

Attribute	Required/ Optional	Description
data-mapping-name	Required	Name of the referenced data mapping.
value-field	Required	Name of a key field in the referenced data mapping that will be used as the source of valid values.
display-value-field-expr	Optional	Select list column SQL expression used to create a display version of the value. Referenced data mapping field names surrounded by curly braces can be used.
value-filter-expr	Optional	SQL where clause condition that allows the selection of a subset of values from the referenced data mapping table that are valid for this field. The expression can use field names in curly braces.

Example

```
<key-field name="IdCounterparty"

display-name="Counterparty ID" description="Counterparty ID" column="IDCPTY" >

<value-type>

<data-mapping-value-type

data-mapping-name="Counterparty" value-field="IdCounterparty"
```

```
display-value-field-expr="{Name}        ({ShortCode})" />
</value-type>
</key-field>
```

A <key-field> can point to any key field in another <data-mapping> tag, except the Record ID. The same functionality as linking to a Record ID can be achieved by specifying a long variable with the matchRecordId attribute set to true as the foreign key. All fields specified as a foreign key in a different data mapping must be defined in a unique group that contains only that single field.

Example of a complex structure

```
<data-mappings>
<!--    SD_BASEVIEW mapping -->
<data-mapping name="BaseView"
table="SD_BASEVIEW" >
<access>
<read   permission="DMadmins" />
<write permission="DMadmins" />
</access>
<key-field name="IdCounterparty"
column="IDCPTY" >
<value-type>
<data-mapping-value-type
data-mapping-name="Counterparty" value-field="IdCounterparty" />
</value-type>
</key-field>
<key-field name="IdProduct" column="IDPRODUCT" >
<value-type>
<data-mapping-value-type
data-mapping-name="Product" value-field="IdProduct" />
</value-type>
</key-field>
</data-mapping>
<!--    SD_COUNTERPARTY mapping -->
<data-mapping name="Counterparty"
table="SD_COUNTERPARTY" >
<access>
```

```
<read   permission="DMadmins" />
<write permission="DMadmins" />
</access>
<key-field name="IdCounterparty"
column="IDCPTY" matchRecordId="true">
<value-type>
<simple-value-type type="Long" />
</value-type>
</key-field>
…
<unique-group name="IdCounterparty_PrimaryKey"
<field-ref field="IdCounterparty" />
</unique-group>
…
</data-mapping>
<!--    SD_PRODUCT mapping -->
<data-mapping name="Product"
table="SD_PRODUCT" >
<access>
<read   permission="DMadmins" />
<write permission="DMadmins" />
</access>
<key-field name="IdProduct "
column="IDPRODUCT" matchRecordId="true">
<value-type>
<simple-value-type type="Long" />
</value-type>
</key-field>
…
<unique-group name="IdProduct_PrimaryKey"
<field-ref field="IdProduct" />
</unique-group>
…
```

```
</data-mapping>

</data-mappings>
```

Create the Database Tables

After you define the data model and configure the data mapping, the next step is to create the database tables.

The DDL (Data Definition Language) creation script is automatically generated by Scrittura after the data model has been configured in `datamapping- config.xml`. This configuration is reloaded in the system using the "SetConfig" process.

This action is available from within the Scrittura administration pages of the user-interface, under the “Database Operation” tab. Once generated, the DDL script can then be copied and pasted by the administrator into a database administration client and run from there in order to create the tables.

For full details on the "SetConfig" process and administration pages, see Scrittura Administration and Run-Time, on page 369.

Configure the User Interface

After the data model is configured and created in Scrittura, the user interface can also be configured to allow manual administration of static data from within Scrittura (such as selection, creation, update, and deletion of records).

User interface configuration is done in a single configuration file, `datamapping-ui-config.xml`, which is located in the /config folder of the Scrittura installation directory.

The following is the basic structure of `datamapping-ui-config.xml.`

```
<?xml version="1.0" encoding="UTF-8"?>

<data-mappings-ui-config home-page-ref="…">

<main-menu>

…

</main-menu>

<pages>

<page>

…

</page>

…

</pages>

</data-mappings-ui-config>
```

Static Data Framework User Interface Layout

The Static Data Framework user interface screens are divided into the following sections:

- **Main Menu.** Lists the different data mappings implemented in the Static Data Framework. The content of a data mapping is displayed in the Content section of the user interface when selected from this menu.

  Displaying the menu is optional, in case you wish to integrate those options directly in the main Scrittura menu.

- **Content.** Displays the actual data. It can be of type Table in order to display a list of records (for example, when a data mapping is selected from the menu) or of type Record in order to display a specific record.

  Both types of content (Table and Record) come with a configurable set of actions (such as create, update, delete, and so on). Each action is configurable such as the next screen to be displayed when the action is triggered.

- **Search**. Allows searching for specific records in the data mappings.

  Searches are contextual to the page currently displayed, which allows refining the record selection when creating a composite record.

  Searches are exact searches, meaning that the results returned will exactly match the criteria. “Like” searches are possible by using the “%” character within the criteria.

  > **NOTE:** The case-sensitivity of the Static Data Framework search is the same as the database case-sensitivity. The case- insensitivity setting for Scrittura trade searches does not apply here.

General Static Data Framework User Interface Configuration

The Static Data Framework user interface is completely configurable. This configuration is done in a single configuration file, datamapping-ui- config.xml, which is located in the /config folder of the Scrittura installation directory.

The following is the basic structure of `datamapping-ui-config.xml`.

```
<?xml version="1.0" encoding="UTF-8"?>

<data-mappings-ui-config home-page-ref="…">

<main-menu>

…

</main-menu>

<pages>

<page>

…

</page>
```

…

```
</pages>

</data-mappings-ui-config>
```

The <data-mappings-ui-config> node is the root node of the datamapping-ui-config.xml configuration file and allows you to define general parameters of the user interface.

The <data-mappings-ui-config> node has the following attributes.

Attribute	Required/ Optional	Description
home-page-ref	Required	Specifies which of the page references defined in the <main- menu> tag will be displayed as the Static Data Framework home page.
date-format	Optional	Specifies the date format to be used throughout the Static Data Framework. Defaults to yyyy-MM-dd if unspecified.
datetime-format	Optional	Specifies the date/time format to be used throughout the Static Data Framework. Defaults to yyyy-MM-dd HH:mm:ss if unspecified.
show-menu	Optional	Specifies whether the menu showing the different data mappings should be displayed in the UI. Default is true.

The date-format and datetime-format attributes are specified by date and time patterns following the standard Java syntax. Refer to the SimpleDateFormat class Javadoc API on the Oracle portal for details.

Examples of common patterns assuming a 24-hour basis

Pattern	Example
dd/MM/yy	10/01/14
dd MMMM yyyy	10 January 2014
HH:mm:ss	17:06:32

The <main-menu> node contains a list of page references which will be displayed as links in the main title bar. Each of these links is specified by an empty child <menu-item> tag that has the following attributes.

Attribute	Required/ Optional	Description
page-ref	Required	Name of the page being referenced.
visible-name	Required	Name displayed to the user in the title bar.
description	Optional	Short description that displays when the mouse hovers over the link.

Example

```
<data-mappings-ui-config home-page-ref="GeneralView">
<main-menu>
<menu-item page-ref="GeneralView"
visible-name="General View" description="Contact database view" />
<menu-item page-ref="CounterpartyList" visible-name="Counterparties"
description="Counterparty details" />
<menu-item page-ref="ProductList"
visible-name="Product" description="Product list" />
</main-menu>
…
</data-mappings-ui-config>
```

The <pages> node takes no attributes and has <page> child nodes. Each page accessible from the user interface is configured within its own <page>. Typically, one page of type Table and one of type Record would be defined for each data mapping, respectively to display the data mapping content and a single record.

Each page displayed on the screen of the user interface is configured within its own <page> tag, a child of <pages>. The following is the general structure of a <page> tag.

```
<page name="GeneralView" type="table"
data-mapping-ref="BaseView" records-per-page="10"
show-bulk-column="true">
<welcome-message>…</welcome-message>
<data>
…
</data>
<child-records>
…
</child-records>
<action-list>
…
</action-list>
<search-criteria>
…
</search-criteria>
</page>
```

The following topics provide further details about the attributes of a <page> node as well as its child nodes:

<page> Attributes

Each page accessible from the user interface is configured within its own <page>. Typically, one page of type Table and one of type Record would be defined for each data mapping, respectively to display the data mapping content and a single record.

A <page> node has the following attributes.

Attribute	Required/ Optional	Description
name	Required	Name of the page.
type	Required	There are two possible types of page: table and record. If table is specified, then the page will display and/or enable manipulation of an entire data mapping. If record is specified then only a single data mapping record will be involved.
data-mapping-ref	Required	Name of the data mapping being used in the page.
records-per-page	Optional	Number of entries listed per page. The default is 20.
show-bulk-column	Optional	Boolean value that specifies whether to display the column of check boxes in the left column. Default is true.

Example

```
<page name="GeneralView" type="table"
data-mapping-ref="BaseView" records-per-page="10"
show-bulk-column="true">
```

<welcome-message> Node

Each page displays a welcome message under the title bar. This is specified as a simple string within the <welcome-message> tag.

```
<welcome-message>Contact Database general view</welcome-message>
```

<data> Node

The <data> node specifies the sources of data for the information to be displayed on the page and has the following structure.

```
<data>
<data-ref … />
…
</data>
```

Each column to be displayed in the screen for this data mapping is defined within its own <data-ref> node.

A <data-ref> node is empty and has the following attributes.

Attribute	Required/ Optional	Description
field-ref	Required	Refers to a specific field within the selected datamapping. ◦ To refer to a field within the specified data mapping only the name of the field itself is needed. ◦ To refer to a foreign field in a table linked by a foreign key, the name of the linked variable (which must be of type Long with the attribute `matchRecordId="true"`) must be specified in square brackets, followed by the name of the foreign data mapping, followed by a full stop, followed by the desired foreign field.
widget	Optional	Specifies whether the item defined by field-ref is read-only, or is viewable by dropdown, checkbox, radio, or text. Default is text.

Example 1: Refer to fields within the same data mapping

```
<data>
<data-ref field-ref="ProductCode" widget="readonly" />
<data-ref field-ref="ProductName" widget="text" />
<data-ref field-ref="ProductFamily" widget="dropdown" />
</data>
```

Example 2: Refer to foreign fields of other data mappings

```
<data>
<data-ref
field-ref ="[CounterpartyID]Counterparty.ShortCode" />
<data-ref
field-ref ="[CounterpartyID]Counterparty.CounterpartyName" />
```

```
<data-ref field-ref = "[ProductID]Product.ProductCode" />
<data-ref field-ref = "[ProductID]Product.ProductName" />
</data>
```

<child-records> Node

If the data mapping being used as a data source for the current page contains foreign links to other data mappings, then it is necessary to specify these in the <child-records> tag in order for these data mappings to specify which screens should be called for the create and update actions.

The `<child-records>` node is an empty node has the following structure.

```
<child-records>
<child-record-mapping … />
…
</child-records>
```

Each foreign link is defined within its own `<child-record-mapping>` child node and has the following attributes.

Attribute	Required/ Optional	Description
data-mapping-ref	Required	Name of the foreign data mapping.
page-ref	Required	Page with which this data mapping should be associated.

Example

```
<child-records>
<child-record-mapping data-mapping-ref="Product"
page-ref="ProductList" />
<child-record-mapping data-mapping-ref="Counterparty"
page-ref="CounterpartyList" />
</child-records>
```

<action-list> Node

The <action-list> node contains the list of actions available for a page and has the following structure.

```
<action-list>
<action … />
…
```

```
</action-list>
```

<action> child nodes define the actions available for a page and have the following attributes.

Attribute	Required/Optional	Description
name	Required	Name of the action. Possible values are create, update, delete, submit, cancel, and search.
page-ref	Required	Page to display after initiating the action.
do-confirm	Optional	Specifies whether to ask for a confirmation before the action is completed. Default is false.
visible-name	Optional	Name displayed for the action in the user interface.

The name attribute has the following possible values.

- **create.** Adds a new data mapping record. Initiating this action will bring up the page specified in page-ref. Represented as a button at the bottom of the screen.
- **update.** Updates a data mapping record. This only works if a single record is selected using a check box. Initiating this action will also bring up a page specified in page-ref. Represented as a button at the bottom of the screen
- **delete.** Deletes whatever data mapping records are selected using the check boxes. Represented as a button at the bottom of the screen.
- **submit.** Carries out a current action. Represented as a button at the bottom of the screen
- **cancel.** Cancels a current action. Represented as a button at the bottom of the screen.
- **search.** Enables search criteria defined by <search-criteria>. Represented as a search bar that displays beneath the welcome message.

<search-criteria> Node

The <search-criteria> node specifies the fields to be used as search criteria. The search criteria displays in the drop-down menu within the search bar. The search action should be defined for that page in order to make use of those search criteria.

The <search-criteria> node has the following structure.

```
<search-criteria>

<data-ref … />

</search-criteria>
```

The <data-ref> child node has the following attribute.

Attribute	Required/ Optional	Description
field-ref	Required	Defines the name of the field to use as search criterion.

Example

```
<search-criteria>
<data-ref field-ref="CounterpartyShortCode" />
</search-criteria>
```

Sample Table and Record Page Configurations

The following are sample configurations for Table and Record pages.

Sample: Table Page

```
<page name="GeneralView" type="table"
data-mapping-ref="BaseView" records-per-page="10"
show-bulk-column="true">
<welcome-message>
Contact Database general view
</welcome-message>
<data>
<data-ref field-ref = "[CounterpartyID]Counterparty.CounterpartyShortCode" />
<data-ref field-ref = "[CounterpartyID]Counterparty.CounterpartyName" />
<data-ref field-ref = "[ProductID]Product.ProductCode" />
<data-ref field-ref = "[ProductID]Product.ProductName" />
</data>
<action-list>
<action name="create"
page-ref="EditGeneralView" do-confirm="true"
visible-name="New" />
<action name="update"
page-ref="EditGeneralView" />
<action name="delete"
page-ref="GeneralView" do-confirm="true"/>
<action name="search"
page-ref="GeneralView" />
</action-list>
<search-criteria>
<data-ref field-ref = "[CounterpartyID]Counterparty.CounterpartyShortCode" />
```

```
<data-ref field-ref = "[CounterpartyID]Counterparty.CounterpartyName" />
<data-ref field-ref = "[ProductID]Product.ProductCode" />
<data-ref field-ref = "[ProductID]Product.ProductName" />
</search-criteria>
</page>
```

Sample: Record Page

```
<page name="EditGeneralView" type="record"
data-mapping-ref="BaseView">
<welcome-message>Contact Database Details</welcome-message>
<data>
<data-ref field-ref = "[CounterpartyID]Counterparty.CounterpartyShortCode"
widget="readonly" />
<data-ref field-ref = "[CounterpartyID]Counterparty.CounterpartyName"
widget="text" />
<data-ref field-ref="[ProductID]Product.ProductCode" />
<data-ref field-ref="[ProductID]Product.ProductName" />
</data>
<child-records>
<child-record-mapping data-mapping-ref="Product"
page-ref="ProductList" />
<child-record-mapping data-mapping-ref="Counterparty"
page-ref="CounterpartyList" />
</child-records>
<action-list>
<action name="submit"
page-ref="GeneralView" do-confirm="false" />
<action name="cancel"
page-ref="EditGeneralView" />
<action name="search"
page-ref="GeneralView" />
</action-list>
<search-criteria>
<data-ref field-ref="CounterpartyShortCode" />
```

```
</search-criteria>

</page>
```

Interaction with the Framework

Programmable interaction with the Static Data Framework depends on the main classes of the Framework. Using these classes, you can perform essential operations like CRUD operations (Create, Read, Update, Delete).

Main Classes, below and Essential Operations, on page 262 provide details on the main classes involved in the programmable interaction process and outline the steps to programmatically perform the main operations on the Static Data Framework.

For more details on the available API, see the Javadoc documentation.

Main Classes

This section lists some of the main classes used by the Static Data Framework, all located in the `com.ipicorp.scrittura.datamapping` package. For full details on the properties and methods available for those classes, refer to the Javadoc documentation.

DataMappingsManager

`DataMappingsManagerService` is the main class of the Static Data Framework. The `DataMappingsManagerService` class functions as a session bean used to perform the essential operations on the framework.

An instance of `DataMappingsManagerService` can be created using the helper class `DataMappingHelper`.

```
DataMappingsManagerService dataMappingsManager

=DataMappingHelper.getDataMappingManager();
```

DataMappingHelper

`DataMappingHelper` is a helper class used to create an instance of the `DataMappingsManagerService` session bean.

It allows retrieving an instance of the `DataMappingsManagerService` session bean using the method `getDataMappingManager()`:

```
public static DataMappingsManagerService getDataMappingManager();
```

DataMapping

The `DataMapping` class is used to represent a data mapping, as defined by the data-mapping tags in the configuration file.

DataMappingRecord

The `DataMappingRecord` class is a collection of key/value pairs that represent a record in a Static Data Framework table.

In addition to user-defined fields, `DataMappingRecord` contains the following built-in fields.

Field	Type	Description
recordid	Long	Record identifier, also the record primary key in the database.
keys	LinkedHashMap	This field objects.
values	LinkedHashMap	This field
createdby	String	Name of the user who created the record.
createdon	Date	Date on which the record was created.
lastModifiedBy	String	Name of the user who modified the record most recently.
lastModifiedOn	Date	Date on which the record was last modified.

The keys object is a map of the fields specified in the configuration file for the data mapping. This map can be retrieved using the operation `getKeys()`.

The keys object must contain a `KeyFieldValue` object for every `<key- field>` tag specified in the configuration file. If it does not, any database operation attempted with this object will fail.

There is no `setKeys(LinkedHashMap keys)` method. Instead, when a new DataMappingRecord object is created, the keys must be added using `addKey(KeyFieldValue key)`. No check against the configuration file is performed at this point.

To modify a key in an existing DataMappingRecord, use `getKeys().get(<fieldname>)`, then modify the KeyFieldValue object.

KeyFieldValue

The `KeyFieldValue` object represents a single key field mapping in the configuration file.

DataMappingLookupRequest

The `DataMappingLookupRequest` class is used to facilitate searches against the database. Search criteria are specified against the different key fields as per the data mapping configuration file.

Exact and Like searches can be achieved using the following methods.

- **Exact Search.** To perform exact searches on the database, use the method `addKeyToMatch (String keyFieldName, Object keyValue)`. This method allows you to search by any data type. Any number of search parameters can be added. If no keys are added, the lookup will return all records that match the request on the specified data mapping. In a large database, this can create a very large list. If necessary, the search can be configured to return only a section at a time using the method `setPage(int pageNumber, int recordsPerPage)`.
- **Like Search.** KeyPatterns can be added using the method `addKeyPatternToMatch(String keyFieldName, String keyValuePattern)`. This method allows you to search for fields using SQL syntax such as wildcard characters '%' or '?'. It can only be used for key fields that have a value type of String.

Essential Operations

Using the main classes on the Static Data Framework, you can perform Create, Read, Update, Delete (CRUD) operations on the Static Data Framework.

These operations can be achieved using the DataMappingManager class, whose methods are listed in this section.

Retrieve a Data Mapping

This operation allows the retrieval of a data mapping object by its name as specified in the `data-mapping` nodes of the configuration file.

Method Signature	`DataMapping getDataMapping`(String `dataMappingName`);
Parameter	`dataMappingName` String containing the name of the required data mapping.
Returns	The `DataMapping` object representing the data mapping table.

Example

```
DataMapping counterPartyTable
= dataMappingsManager.getDataMapping(“Counterparty”);
```

The following statement returns the string “`List of counterparties`”.

```
counterPartyTable.getDescription();
```

The following statement returns the list of roles with write permission on the counterparty table.

```
counterPartyTable.getWritePermissions();
```

Record Lookup

This operation queries the database using a lookup request object. This operation always returns a list, even if only one result is possible.

Method Signature	List lookup (String `dataMappingName`, `DataMappingLookupRequest request`)

Parameter	`dataMappingName`
	String containing the name of the database table.
	`request`
	The lookup query.

Returns	The list of Data Mapping records (of `DataMappingRecord` type) that matches the request.

Example

```
DataMappingLookupRequest request = new DataMappingLookupRequest();
request.addKeyPatternToMatch(“CounterpartyShortCode”, “%a%”);
```

The following statement returns all the records in the counterparty table which have the letter A somewhere in the `CounterPartyShortCode` field. Note that SQL wildcards like ‘%’ and ‘?’ can be used here.

```
List<DataMappingRecord> results = dataMappingsManager lookup(“Counterparty”, request);
```

The following statement returns the string containing the shortcode of the first counterparty record.

```
DataMappingRecord first = results.get(0); first.getKeys().get(“
CounterpartyShortCode”)
```

Count Records

This operation counts the number of records in the database that match a particular lookup request.

Method Signature	`int count (String dataMappingName, DataMappingLookupRequest request)`

Parameter	`dataMappingName`
	String containing the name of the database table.
	`request`
	The lookup query.

Returns	The number of records that match the request.

Example

```
DataMappingLookupRequest request = new DataMappingLookupRequest();
request.addKeyPatternToMatch(“CounterpartyShortCode”, “%a%”);
```

The following statement returns the number of records in the counterparty table which have the letter "a" somewhere in the CounterpartyShortCode field. Note that SQL wildcards like ‘%’ and ‘?’ can be used here.

```
dataMappingsManager.count(“Counterparty”, request);
```

Select a Specific Record

This operation retrieves a specific record from the database. For this operation, you must provide the ID of the record. Because the record ID is unique, this method always returns a single object.

Method Signature	`DataMappingRecord getRecord (String dataMapingName, Long recordId)`
Parameter	`dataMappingName`

	String containing the name of the database table. `recordId` The record to retrieve.
Returns	The record with the specified ID.

Example

```
DataMappingRecord record = dataMappingsManager.getRecord (“Counterparty”, 1);
```

The following statement returns the string containing the shortcode of this counterparty.

```
record.getKeys().get(“ CounterpartyShortCode”)
```

Insert a Record

This operation inserts a new record into the database. The DataMappingRecord object contains all the record information to add with the exception of the ID, which is set when the database is updated.

If you have only a list of values, you must create a new DataMappingRecord and add the values to it, then use this object to insert the record. All key fields with the property optional set to false in the configuration file should have a matching value in the new record’s key list.

Any constraints listed in the containing data-mapping must be respected. If constraints are violated, the insert operation will fail and an exception will be thrown.

Method Signature	`void addRecord (String dataMappingName, DataMappingRecord record)`

Parameter	`dataMappingName`
	String containing the name of the database table.
	`record`
	The record to add.

Returns	This method does not return anything, but an exception will be thrown if the insert operation fails.

Example

```
DataMappingRecord record = new DataMappingRecord(); record.addKey(new KeyFieldValue
(“ShortCode”, “a shortcode”)); record.addKey(new KeyFieldValue(“Name”,“example
counterparty”)); record.addValue(“IsReadOnly”, false); dataMappingsManager.addRecord
(“Counterparty”, record);
```

The ID of the record is set when the record is successfully added to the database.

Update Record Key Fields

This operation is used to update the fields of a record already existing in the database. It does not use a DataMappingRecord object but requires a list of key and value fields to update. To update a

DataMappingRecord object, you must extract the information from it, then perform this operation using the ID from the record object. Only key fields that have changes are required.

All constraints specified on the configuration file must be respected.

Method Signature	`void updateRecordFields (String dataMappingsName,`
	`Long recordId,`
	`List<KeyFieldValue> keyFields,`
	`Map<String, Object> valueFields`

Parameter	`dataMappingName`
	String containing the name of the database table.
	`recordId`
	The ID of the record to update.
	`keyFields`
	The fields to be modified.
	`valueFields`
	The values to be modified.

Returns	This method does not return anything, but an exception will be thrown if the update operation fails.

Example

```
List values = newLinkedList(); values.add(new KeyFieldValue(“ShortCode”,
“a different shortcode”; dataMappingsManager.updateRecordFields(“Counterparty”, 1,
values, null);
```

Delete Record

This operation removes a specific record from the database. You must provide the ID of the record to delete.

The lookup request method can be used to find the record ID.

Method Signature	`void deleteRecord (String dataMappingName,` `Long recordId)`

Parameter	`dataMappingName`
	String containing the name of the database table.
	`recordId`
	The ID of the record to update.

Returns	This method does not return anything, but an exception will be thrown if the update operation fails.

Example

```
dataMappingsManager.deleteRecord(“Counterparty”, 1);
```

Chapter 10: Scrittura Utility Modules

This section describes the main utility modules available in Scrittura, the job scheduler and archiving solution.

This section contains the following topics:

Job Scheduler

The Job Scheduler in Scrittura allows tasks to be scheduled and run automatically. Some tasks are packaged with Scrittura. Custom jobs can also be configured. Jobs can be set to run synchronously or asynchronously.

Job Scheduler Configuration

The Scrittura scheduler uses the UNIX 'CRON' syntax to schedule the execution of tasks (also called cronjobs). These tasks could include the execution of reports, a bulk 'push' forward in the workflow, or any other logic coded within Java classes.

The scheduler is configured through the scheduler.xml file and has the following structure.

```
<scheduler>
<users>
<user .../>
</users>
<jobs>
<job ... >
<schedule .../>
<param .../>
...
</job>
...
</jobs>
</scheduler>
```

The root node of `scheduler.xml` is `<scheduler>`, which takes no attributes and accepts child nodes discussed in the following topics:

- <users> Node, below
- <jobs> Node, on the next page

Example `scheduler.xml` file that executes

```
com.ipicorp.scrittura.scheduler.jobs.SendReport job

<scheduler>

<users>

<user username="admin" password="admin" />

</users>

<jobs>

<job name="sendReport" class="com.ipicorp.scrittura.scheduler

.jobs.SendReport"

runAs="admin">

<!-- every hour, during business hours -->

<schedule min="0"

hour="9-18" day="*" month="*" weekday="1-5" />

<param key="reportClass" value="com.ipicorp.scrittura

.util.schedule.TestingReport"/>

<param key="format" value="CSV"/>

<param key="emailFrom" value="fromEmail@company.com"/>

<param key="emailTo" value="toEmail@company.com"/>

<param key="emailCc" value="ccEmail@company.com"/>

<param key="emailBcc" value="bccEmail@companyp.com"/>

<param key="emailSubject" value="Scrittura Report"/>

<param key="emailBody" value="Example Report"/>

</job>

</jobs>

</scheduler>
```

> **NOTE:** The <param> elements are specific to the scheduled class.

<users> Node

Users that are used to run the cronjobs are defined under the <users> node, each of them being specified as a `<user>` child node of `<users>`.

A `<user>` node is empty and takes mandatory attributes, username and password, which define the user's credential. Such users must also be defined in the security realm of the application server.

Passwords can be encrypted using the startup-config.xml configuration file. For more information, see SetConfig Process Configuration, on page 373.

<jobs> Node

Cronjobs are defined under the <jobs> node, each of them being specified as a <job> child node of <jobs>.

The <jobs> node has the following attribute.

Attribute	Required/ Optional	Description
run-async	Optional	Specifies whether the cronjobs should run synchronously or asynchronously. By default, cronjobs run synchronously.

A <job> child node has the following attributes.

Attribute	Required/ Optional	Description
name	Required	Defines a unique name for this job.
disabled	Optional	Boolean value that lets you disable the automatic schedule of a cronjob in order to only run it manually. Default: false
class	Required	Specifies the fully qualified name of the Java class run by the cronjob.
runAs	Required	Defines the user used to run the cronjob. The value must be one of the users defined under the <users> node.

A <job> child node has the following child nodes.

Child Node	Required/ Optional	Description
schedule	Required	Specifies when to run this job using the 'CRON' syntax, using its mandatory day, hour, min, month, and weekday attributes.
param	Optional	Specifies a parameter that is used by the cronjob, using its key and value parameters. A job element can have more than one param child node.

Create a Scheduled Task

Custom `cronjobs` can be added to the client application as Java concrete classes that subclass the `com.ipicorp.scrittura.util.CronJob` abstract class and override its perform`Task()` method.

Once the `cronjob` Java implementation is complete, add the job to the scheduler.xml and specify schedule time and credentials under which it should run.

The class `com.ipicorp.scrittura.scheduler.LoadCronThread` must also be added to the list of startup classes in `scrittura-config.xml`.

Cronjobs Running in Synchronous Mode

By default, all jobs run synchronously in a single thread through the Job Scheduler.

The scheduler checks for jobs at one minute intervals. If a set of jobs takes more than one minute to run, the scheduler checks when the thread becomes available.

This may become an issue if, for example, job A runs every five minutes and job B takes 20 minutes to run. While job B is running, job A should have run four times but cannot because job B is using the thread. When job B completes, job A will be immediately queued to run, but it will run only once— not four times.

Cronjobs Running in Asynchronous Mode

Setting the `run-async=true` option for the Jobs tag changes the behavior of cronjobs to use a JMS-based implementation. With this setting implemented, Scrittura sends a message to a JMS queue each time a job is scheduled to run, and the job runs when the message is received. This allows you to run multiple jobs in parallel.

Considerations for running cronjobs in asynchronous mode:

- One drawback of the asynchronous method is that if a job takes a long time to execute and is scheduled too often, it may start backing up in the JMS queue and may make cronjobs run late. This should not be an issue in general, but could happen with broken jobs.
- Asynchronous jobs are not guaranteed to run when they are scheduled.

  For example, this could occur of a very large number of cronjobs are running and some of them take an excessively long time to execute, or they are not user friendly in that they require a large update across the database. In this case, it is a good idea to ensure that the relevant jobs are only run at convenient time (if current time = working time, abort).
- When using asynchronous mode, a new JMS queue has to be defined. The JMS queue JNDI name should be set to:

  ```
  cron_jobs (Store enabled, redelivery limit == 1)
  ```

  By default, there are three listeners on this queue. This can be changed in the deployment descriptor.

Built in Scheduled Tasks

Scrittura's scheduler functionality includes the following built-in tasks that can be scheduled.

- SendSaveSearch
- SendReport
- ClearActivity

- WriteStatisticLog
- WorkflowStatus
- AbstractArchiveCronJob
- QueueArchiveCronJob

Email Search Results: SendSavedSearch

The `com.ipicorp.scrittura.scheduler.jobs.`SendSavedSearch class runs a public or user-specific saved search and sends it through email as a PDF or a CSV (comma-separated values) file.

The class expects the following variables to be set in the `scheduler.xml` file.

Variable	Required/ Optional	Description
searchType	Required	Type of search, user or public.
searchName	Required	Name of the search.
searchCategory	Required	Category of the search, as specified in scrittura-config.xml. Valid only when searchType is public.
searchFormat	Required	Format of the search results, PDF or CSV.
emailFrom	Required	The From: header of the email.
emailTo	Required	The To: header of the email. Comma-separated values
emailCc	Optional	The Cc: header of the email. Comma-separated values
emailBcc	Optional	The Bcc: header of the email. Comma-separated values
emailSubject	Required	The Subject: header of the email.
emailBody	Required	The body of the email.

The results of the search include the workitems that the report would display if it had been run through the GUI, therefore only include those workitems the user is permitted to see.

Email Report Results: SendReport

The `com.ipicorp.scrittura.scheduler.jobs.SendReport` class sends report results through email as a PDF or a CSV file. This class supports only reports that implement the `com.ipicorp.scrittura.util.StandaloneReport` interface.

The class expects the following variables to be set in the `scheduler.xml` file.

Variable	Required/ Optional	Description

reportClass	Required	The name of the report class to run. This class must implement the interface
		`com.ipicorp.scrittura.util`
		`.StandaloneReport.`

format	Required	Format of the report, PDF or CSV.
emailFrom	Required	The FROM: header of the email.
emailTo	Required	The TO: header of the email. Comma-separated values
emailCc	Optional	The CC: header of the email. Comma-separated values
emailBcc	Optional	The BCC: header of the email. Comma-separated values
emailBody	Required	The body of the email.

Push Forward Workitems: ClearActivity

The `com.ipicorp.scrittura.scheduler.jobs`.ClearActvity class checks each item in a given activity to see if it has been there for more than a specified amount of time. If an item has been there for more than the specified time, it is automatically forwarded. This class optionally sets a variable to true to mark that the item was moved. This variable could then be used in a workflow definition to route the workitem to the next appropriate activity.

The class expects the following variables to be set in the `scheduler.xml` file.

Variable	Required/ Optional	Description
activity	Required	Specifies the workflow process and activity to monitor. Separate the process and activity by a period. For example, Scrittura.Review.
maxIdleMinutes	Required	Specifies how long an item must be waiting before it can be marked for forwarding.
variableToSet	Optional	Specifies a PI variable that will be set to true before the item is forwarded. This PI variable must be defines in commonvars (or all appropriate Product Definition files).

Example

The `ClearActvity` class is set to run once a day at 5pm. Items in activity X are scheduled to be forwarded using `ClearActivity` after sitting in an activity for 2 hours. If an item arrives in activity X at 4pm, the item is not forwarded out at 5pm since it is only an hour old. Instead, it is forwarded the next time `ClearActivity` runs, which in this case is 5pm the next day.

A business application could be in an environment where multiple versions of a trade are amended many times during a day, each creating a new internal document version in Scrittura. `ClearActivity` could be scheduled to run against a "waiting for release" manual activity to prevent a document from being dispatched to a customer unless it has remained unamended for a specified number of minutes.

The variable `maxIdleMinutes` compares the present time to the time at which the workitem arrived in the queue. A "save" without forwarding of the workitem does not reset this count.

Workflow Statistics Logging

The `com.ipicorp.scrittura.scheduler.jobs`.WriteStatisticLog class enables Workflow statistics to be logged. See Workflow Reporting Module, on page 101.

The class expects the following variable to be set in the `scheduler.xml` file.

Variable	Required/ Optional	Description
value	Required	The absolute path to the directory where the logs are stored. For example, `/opt/scrittura/logs`

Example

```
<job name="emailWorkflowStatus" class="com.ipicorp.scrittura.scheduler
.jobs.WorkflowStatus"
runAs="admin">
<schedule min="0"
hour="*"
day="*" month="*" weekday="*" />
<param key="path"
value="@scrittura.home@/logs"/>
</job>
```

Workflow Status Monitor

The `com.ipicorp.scrittura.scheduler.jobs`.WorkflowStatus class enables workflow activities to be monitored directly from the "Workflow Status" screen. See Scrittura Administration and Run-Time, on page 369.

The class expects no specific variables to be set in the `scheduler.xml` file.

Example

```
<job name="emailWorkflowStatus"
class="com.ipicorp.scrittura.scheduler.jobs.WorkflowStatus"
runAs="admin">
<schedule min="0"
```

```
hour="*" day="*"
month="*"
weekday="*" />
</job>
```

Archiving Cronjobs

Scrittura provides an abstract Archiving cronjob, `com.ipicorp.scrittura.scheduler.jobs.AbstractArchiveCronJob`. This class can be extended to provide customized automated archiving.

A simple implementation, `com.ipicorp.scrittura.scheduler.jobs.QueueArchiveCronJob`, is also provided. For details, see Automated Archiving, on page 279.

Archiving

The Archiving module allows moving trades and documents offline once they are no longer actively used. The implementation supports the resurrection of trades and documents if they need to be accessed or changed. It is also possible to browse the Archive without resurrecting the data to the database.

Archived data is stored as a ZIP file containing the documents and two XML files containing a snapshot of the database data for the trade. This ZIP file can be stored offline on tape or DVD, on a near-line database or file system, or on the Scrittura database. Scrittura natively includes a filesystem solution.

If stored offline, browsing the Archive data requires that a request for this Archive be sent to the technical staff so that the data can be made available to Scrittura. On a near-line and online database, the system is able to resurrect trades on demand.

Search variables can be defined for archived trades to allow searches on the Archive.

Archiving Configuration

The options described in this section must be configured in `scrittura- config.xml`, located under the Scrittura configuration folder.

Base Archiving Parameters

All archiving configurations are located under the <archive> node in the `scrittura-config.xml` configuration file.

The `<archive>` node has the following attributes.

Attribute	Description	Possible Values
enabled	When set to true, automatic archiving is enabled.	`true` `false (default)`

storageClassTool	Classname of the handler for archive loading and unloading. The filesystem archiving implementation is natively provided by Scrittura, `com.ipicorp.scrittura.archiving.FileSystem`. Custom storage classtools can be developed by implementing the `com.ipicorp.scrittura.archiving.StorageClassTo ol` interface.	String

storageAttributes	A semi-colon separated list of attributes for the chosen `storageClassTool.`
	The following are the parameters for the filesystem implementation:
	◦ `inboundPath`. Path to the directory that contains the archives for browsing and resurrecting.
	◦ `outboundPath`. Path to the directory that contains the generated archives (from where they should be moved to tape).
		String
		inboundPath default:
		`${scrittura.home}/archivesIn`
		`outboundPath default:`
		`${scrittura.home}/archivesOut`

documentSelectorClassTool	Classname of the handler for document selection.	String Default: `com.ipicorp.scrittura.archiving.TradeFolder` (one folder per trade) Create your own by implementing: `com.ipicorp.scrittura.archiving.DocumentSelectorClassTool`
backfillerClassTool	Class name of the back-filler	String Default: empty (no back filler) Create your own by implementing: `com.ipicorp.scrittura.archiving.BackfillerClas sTool`
docmgr-filesystem-mapping	Defines the naming convention to use for storage on the filesystem. For	String

	filesystem, see `docmgr-external-config.xml`.	
version-prefix	Defines the naming convention for differentiating document versions to use for storage on the filesystem. For filesystem, see `docmgr-external-config.xml`.	String
delete-zip-files	When set to true, the archive ZIP file is deleted when the trade is restored in the live system.	true false (default)
roles	Roles that define permissions for manual archiving	String none, all, or a comma- separated list of roles
useExternalArchiving	Boolean attribute used when DocManager is integrated with an external DMS. If set to true, archiving of documents is delegated to the external DMS.	true false (default)
forbidden-chars replacement-chars	Strings listing the forbidden characters and their replacement to generate file paths.	String
missingArchiveColumnPlace holder	When performing a search, specifies the display of a search result column for archived trades when that column is defined in the FA Tables but not in the Archiving table.	String

The `<archive>` node has the following child nodes.

Child Element	Description
archive-var	Defines a variable that can be searched on for archived trades. Multiple archive-var elements can be defined; each must contain a single name attribute that lists the internal variable name of the variable that will be searchable. For details, see Archive Search Setting, on the next page.
archive-activities	Allows the customization of the activities used by the archiving workflow. For details, see Customizing the Archiving Workflow Activities, on page 280.

Archive Search Setting

You can define a list of variables that will be available for searches on archived variables. The names must correspond to the internal names of the variables.

If you change the list of variables, you must run the `setConfig` process and regenerate the archive tables, and then migrate the current table to the new definition.

Default Archiving Implementation

The default implementation for archiving is trade binding, meaning that it collects all the documents contained in the trade's folder.

For implementations of Scrittura that do not have a unique folder per trade, or have more complicated settings, this can be overridden by providing another document selector. To achieve this, just change the setting in the configuration and implement the `com.ipicorp.scrittura.archiving.DocumentSelectorClassTool interface`.

Archive Processes Summary

The following summaries describe the Archive processes that can be performed.

Select trades to be archived (manual/automatic)

Trades are moved to an archive workflow by a classtool, a cronjob, or manually.

- Trades are serialized into a zip file each. The zip contains XML versions of the trades and the trades' documents.
- The zip is stored on the file system (or near-line).
- Trades are deleted from the database.
- Some (customizable) variables from the trade are kept in the system to allow searches/reconciliations.

Perform Archived trades search

The Trade Search screen offers the option of searching trades that have been archived by a few selection criteria, such as CRID, PIID, or custom fields.

Perform Trade lookup

Trade documents and data can be viewed when the trade archive is available on the file system without having to resurrect them.

Trades can be moved from tape to the file system after being archived.

Perform Trade resurrection

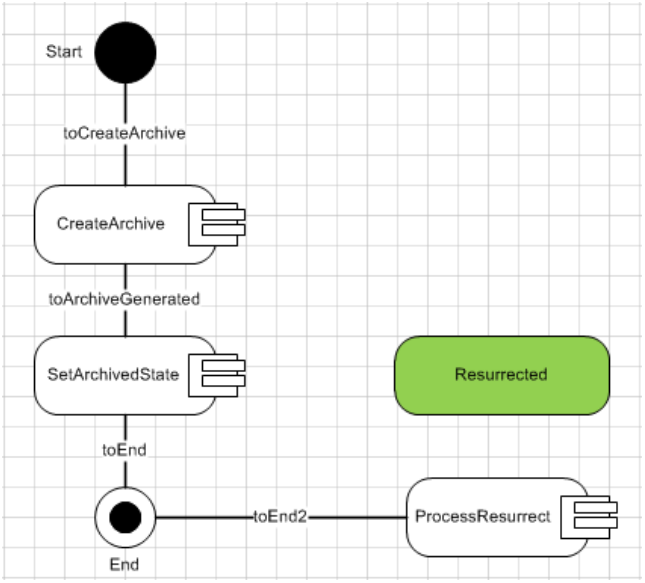
Trades can be automatically or manually moved back into the system from their archived state if their ZIP archive is available on the file system.

Archiving Workflow

This section describes the steps required to use the archive module. These steps can be manual or automated.

The archiving process is based on a separate archiving workflow, `archiving.vsd`, which performs the necessary steps to archive a trade.

The following illustration depicts the archiving workflow.



The archiving process consists of two stages. The first stage generates the archive for the trade (`CreateArchive`). The second stage marks the trade as archived and removes it from the Scrittura database (`SetArchiveState`).

`CreateArchive` generates the trade archive output as a compressed (ZIP) file. The native Scrittura implementation is to store the ZIP archives on the filesystem in the output folder specified by the configuration. The generated file can be kept in this location if it is the storage medium or moved to another location (other disk, tape, DVD). This step can be automated or manual.

Custom implementations can be designed to store the ZIP files elsewhere, such as on the same database (the benefit being to remove the number of items in the indexed tables for performance), or to remove the ZIP to a near- line database with cheaper storage and backup (trades can be archived once a month and the database can be backed up as needed).

The `ProcessResurrect` activity restores trades to their previous location, where possible. If the trade cannot be restored to the previous location, it is added to the Resurrected queue. For details, see Resurrecting a Trade, on the next page.

Manual Archiving

The Scrittura user interface lets you archive a trade in different manners. In all cases, those trades are moved to the "Start" activity of the archiving workflow.

- From the Workflow status screen.

  Trades should be moved to the "Start" step of the "Archiving" workflow.

- Using the trade search.

  When a search is performed, an **Archive** link is available in the search results page. This link uses the Scrittura "arch" event, implemented by the ManualArchiveEvent class. This can be accessed using a uri, such as `http://localhost:7001/scrittura?e=arch&i=[crid]`. The usage of this event is limited to users with the roles defined in the archive configurations.

Search result screens display both archived and live trades. When you click an archived trade, an off-line summary of the trade is displayed (as long as the ZIP archive is accessible to Scrittura). This screen offers the possibility to view the trade documents, annotations, history, and product dump. The trade is not restored in the live system when you click it; therefore no other action is possible.

Automated Archiving

Cronjobs can be used to automatically move trades to the archive workflow without administrator intervention.

An abstract `Cronjob, com.ipicorp.scrittura.scheduler.jobs.AbstractArchiveCronJob`, is supplied with Scrittura and archives trades with `CommonReferenceIDs` that are supplied by the custom implementation of the abstract `getCRIDListToArchive()` method.

An implementation of this is also provided in the form of the `com.ipicorp.scrittura.scheduler.jobs.QueueArchiveCronJob` class which archives any trades remain within a specified queue for a defined length of time. This cronjob has the following parameters:

- workflow defines the workflow to monitor.
- queue defines the activity to monitor within the workflow.
- `daysInQueue` defines the time after which a trade in that queue should be archived.

Resurrecting a Trade

A trade can be resurrected at the request of a user or if an amendment is entered into the system.

In order to automatically resurrect a trade when an amendment comes in, the following classtool should be added to the MessageProcessing workflow as soon as the CommonReferenceID of the incoming trade is identified:

`com.ipicorp.scrittura.archiving.CheckArchiveClassTool`

Trade resurrection can also be accomplished manually from within the trade search. An Unarchive link is available on the search results page, and uses the Scrittura "unarch" event, implemented by the `ManualArchiveEvent` class . Thi can be accessed using a uri such as `http://localhost:7001/scrittura?e=unarch&i=[crid].`

By default, trades are restored to their original location. If for any reason this is not possible (such as, the original location no longer exists), the trades are moved to the `Archiving.Resurrected` activity. Audit records and DocManager files retain their original dates.

Customizing the Archiving Workflow Activities

By default,

- The archive activity is set to `Archiving.Start`.
- The unarchive activity is set to `Archiving.ProcessResurrect`.
- The fallback unarchive activity is `Archiving.Resurrected.`

These can be altered by editing the `archive-activity`, `unarchive- activity`, and `fallback-unarchive-activity` sub-elements of the `archive-activities` element within the archiving section of `scrittura-config.xml`.

Each of these elements has a `workflow-id` and an `activity-id` attribute in order to define the workflow entry points. Samples of `scrittura- config.xml` file are provided with the Scrittura distribution.

Evolving Systems

Because Scrittura configurations evolve, it might be necessary to backfill variables to a trade before resurrecting it. The archive contains a timestamp of the archiving time and a Backfiller tool can be configured to pre-process the archive before resurrecting the trade.

Archiving Caveats

Note the following:

- `DOCMGR_RESOURCEACCESS` is not archived because it is tied to the current configuration of DocManager and its size is minimal.
- Some audit information is not bound to a trade (all the message tickets info), and therefore cannot be archived.

  They can be removed using the reconciliation features from within the workflow status screen.

Chapter 11: Message Processing Workflow

The topics in this section describe important steps and components involved in the Message Processing workflow, their configuration, and interfaces.

This section contains the following topics:

Generic XML Parser

The Generic XML Parser is used to parse any XML schema and map the schema against Message Ticket variables in a fully configurable manner.

A parser instance is completely reusable and has internal configuration caches to improve its performance. An instance can be created once and used multiple times as needed. Parser instances are thread-safe.

Parser instances are based on XPath-like expressions placed in CSV configuration files, located under the `/xmlParserConfig` directory in the Scrittura live folder. Any number of instances can be created and used by the system.

This section details the steps to configure and use the Generic XML parser.

Parser Instance Configuration

The Generic XML Parser configuration file must be in standard CSV format with each row defining a rule for XML elements matching certain conditions. The columns for each row should match the following information.

Column	Description
1	Matching element as an XPath-like expression
2	Name of the Message Ticket variable to be set
others	Chain of additional processing instructions that are applied to the value determined by the parser before it is set into the variable

Rows can be commented out in two ways:

- Leaving the first cell of the row empty
- Placing a hash mark (#) at the beginning of the row

In addition to the element instructions rows, options can be defined anywhere in the configuration file. Option rows let you adjust certain parser parameters. Options rows are specified with the following attributes.

- The text options must appear in the row's first cell.
- From the row's second cell on, the actual options are specified, one option per cell. An option is specified using the option name followed by an equal sign (=) and the option's value.

Available options include the following.

- base. Prefix added to each XPath-like expression in the element instructions rows that follow the option row. This option can appear anywhere in the configuration file and can appear multiple times to set prefixes for blocks of matching rules.

Configuration File Location

Parser configuration files are located under the `/xmlParserConfig` directory in the Scrittura live folder.

Different parser configuration files can be created for different types of messages. For a given message type, the name of the parser configuration file to apply is the message root note with the `extension.csv`. If no parser configuration files can be found, the default configuration file, `defaultConfig.csv`, can be defined and used instead.

Matching Expressions

The Generic XML parser uses XPath-like syntax to match XML elements and attributes to message variables. The following are examples of valid expressions supported by the parser.

Column	Description
/trade/trade-date	Uses the content of the trade-date element as the variable value.
/trade/counterparty/@name	Uses the value of the name attribute of the counterparty element.
/trade/cptys/cpty[1]/shortName	Uses the content of the shortName sub-element of the cpty element, which is the first child of the `cpty's` element
/trade/cptys/cpty{position()<=3} /shortName	Matches the `shortName` sub-elements of the first three cpty children of the cpty's element.
/trade/cptys/cpty[@type]	Matches cpty elements that have type attributes.
/trade/cptys/cpty[@type='DIRECT']	Matches cpty elements whose type attribute is DIRECT.
/trade/cptys/cpty/shortName[.]	Matches `shortName` elements that have non-empty content.
/trade/cptys/cpty/shortName [.='SAIL']	Matches `shortName` elements whose content is `SAIL`.

/trade/instrument [starts-with(., 'Physical')]	Uses the content of the instrument element if it starts with Physical.
/trade/instrument [not(contains(@type, 'Gold'))]	Uses the content of the instrument element if it does not contain the substring `Gold`.
/trade/payments//payment/@date	Uses the date attribute of any payment element, which is a descendent of the payments element, whether it is direct or not. **CAUTION:** Be sure to include the double-slash.
/trade/contacts/contact[2] [@type='FAX']	Matches the second contact sub-element of the contacts element if the type attribute of the contact sub-element is FAX.

Processing Instructions

Generally, a rule is used to extract a value from an XML element or an attribute and save it to a message variable. However, the extracted value can be pre-processed using processing instructions before it is used for the variable.

The processing instructions are chained together, with each instruction processing the value returned by the previous instruction. Any processor in the chain can decide that the variable should not be set using this rule and can return a null value. In this case, no other processors in the chain will run and the whole rule is skipped.

The following standard processors are supported.

Processor	Description
upper()	Makes the value upper-case.
lower()	Makes the value lower-case.
append(string)	Appends the value with the specified string.
prepend(string)	Prepends the value with the specified string.
string()	Converts the value to a string. This is useful in a processing instructions chain when a previous processor returns a non- string value, but a string must be saved as the variable value.
substring(begin, end) substring (begin)	Takes the substring of the value.
static(string)	Assigns the specified string to the variable regardless of the value extracted by the parser. This is useful in situations when a variable must be set if a certain element or

	attribute is present. The string can have embedded “{...}” constructs with EL expressions, allowing dynamically calculated values.
position()	Uses the element’s position among its siblings (starting from 1) as the value.
count()	Counts the number of matching elements and use the current count as the value.

Note the difference between `position()` and `count()`, as shown in the following example for the matching pattern “/a/b/c[@n=’a’]”.

```
<a>
<b>
<c n=”a”/> <!-- position() = 1, count() = 1 -->
<c n=”a”/> <!-- position() = 2, count() = 2 -->
</b>
<b>
<c n=”a”/> <!-- position() = 1, count() = 3 -->
<c n=”b”/> <!-- position() = 2, count() n/a -->
<c n=”a”/> <!-- position() = 3, count() = 4 -->
</b>
</a>
```

The `position()` processor gives the number of the element among its immediate siblings and does not take into account the matching element predicate. In certain situations, `position()` and `count()` are interchangeable.

Processing Instructions Order

Instructions are processed from top to bottom. Multiple rows can match the same element. In this case, processing instructions generally are executed in the order in which they appear in the configuration file.

Example

If the element contact with the attribute type = EMAIL is encountered by the parser, and the parser uses the following configuration,

```
/trade/contacts/contact/fullName,fullName,I1,I2
/trade/contacts/contact[@type=’EMAIL’]/fullName,emailName,I3,I4
```

the parser will execute instructions in the following order:

1. Instruction I1
2. Instruction I2

3. Set fullName variable
4. Instruction I3
5. Instruction I4
6. Set the emailName variable

However, the parser groups rules with exactly the same matching expressions together and appends all instructions to one list. For example, if the configuration is,

```
/trade/contacts/contact[@type='EMAIL']/fullName,emailName,I3,I4
/trade/contacts/contact/fullName,fullName,I1,I2
/trade/contacts/contact[@type='EMAIL']/fullName,emailMarker,I5,
```

the parser will execute instructions in the following order:

1. Instruction I3
2. Instruction I4
3. Set emailName variable
4. Instruction I5
5. Set emailMarker variable
6. Instruction I1
7. Instruction I2
8. Set fullName variable

The third rule is executed before the second because it uses exactly the same matching expression as the first rule and is therefore grouped with the first rule.

Use of EL Expressions in Processing Instructions

The variable names, as well as the argument of the `static()` processing instructions, can contain EL expressions in curly braces. In addition to EL expressions, the following functions are available.

Function	Description
content()	The element content before it is processed by any processing instructions.
attribute(name)	The element attribute with the specified name.
qname()	The element name.

The following functions can be used only in the `static()` value processor, but not in variable names.

Processor	Description
current()	The current value passed to the processor.

skip()	Special function which, if invoked, causes the whole rule to be skipped and the variable not to be set. Usually, this function is used with the standard EL `if()` function. This allows skipping the rule based on certain EL conditions.

Array Handling

Using expressions allows you to create complex and flexible parser logic. The following example uses this logic to create an array. After the message is completely processed, the contactsSize variable will contain the size of the contacts arrays.

```
/trade/contacts/contact,contactsSize,count()

/trade/contacts/contact/fullName,contactFullName[{contactsSize}]

/trade/contacts/contact/email,contactEmail[{contactsSize}],lower()
```

Alternatively, one and two dimensional arrays can be handled easily by the XML parser when capturing all values of a multi-instantiated node is required.

For this purpose, assign the PI array variable (with square brackets) to the node.

Example

The incoming XML contains the following:

```
<FILE>

<ITEM>

<rates>

<rate>1.11</rate>

<rate>2.22</rate>

<rate>3.33</rate>

<rate>4.44</rate>

</rates>

</ITEM>

</FILE>
```

In order to parse and retrieve all the values of the <rate> nodes in a rate[]

PI array variable, the CSV parser would contain the following line:

```
/FILE/ITEM/rates/rate,rate[],,
```

It is possible to introduce temporary variables that can be used in expressions, but do not appear in the parsed variables map. The name of the variable in the second column must be prefixed with a hash to make it a local variable:

```
/trade/contacts/contact,#contactsSize,count()

/trade/contacts/contact/fullName,contactFullName[{contactsSize}]

/trade/contacts/contact/email,contactEmail[{contactsSize}],lower()
```

The end result will not contain the `contactsSize` variable.

> **NOTE:** Arrays are populated by the XML Parser using numeric indexes starting from 1.

Integration with Scrittura

The Generic XML parser can be integrated with the Scrittura application using the same process used to integrate any parser—by adding this parser as a `message-type` node in `scrittura-config.xml`. For full details, see Scrittura Configuration, on page 27.

Standalone Trade Message Parsing

The fully qualified name of the Scrittura parser class is

com.ipicorp.scrittura.messages.GenericXMLParser.

This class should be added to the general Scrittura configuration in the `scrittura-config.xml` file located under the config repository in the Scrittura live folder. A new `<message-type>` tag will be added under the root tag `<scrittura-config>`, adding this class to the list of available parsers.

```
<scrittura-config ...>

...

<message-type name=”XMLMessage”

class=”com.ipicorp.scrittura.messages.GenericXMLParser” />

...

</scrittura-config>
```

Structured Product Message Parsing

Structured Products may reach Scrittura in different ways; the following are the most common:

- Components are sent individually in separate XML messages
- All components are sent together as a single XML message

The first method falls under the common parser use case, while the second method requires the incoming XML message to be split in order to generate one `MessageTicket` per component.

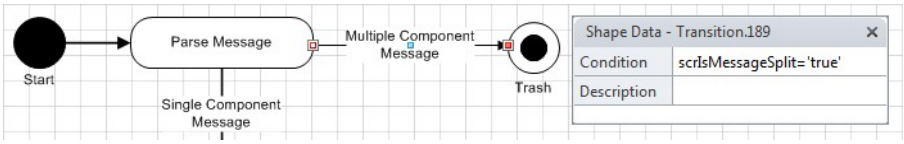
Structured Products use their own configuration file, `structured-product- config.xml`, located in the config directory of the Scrittura live folder.

In this configuration, the names of nodes representing independent components can be defined under the `<parser-component-xpath>` element. The incoming message is then split into as many messages as the number of nodes that are re-injected at the start of the `MessageProcessing` workflow, whereas the initial message should be discarded.

A temporary `MessageTicket` is created for the initial aggregated XML message as well. This MessageTicket contains a single property, `scrIsMessageSplit`, which is set to true. Based on the value of this property, the `MessageTicket` can be routed to the end of the `MessageProcessing` workflow to be destroyed or stored.

It is advised that you do not create a Product Instance (PI) in the main workflow for the initial aggregated XML message.

The following illustration shows an example of the MessageProcessing workflow that handles aggregated messages.



The fully qualified name of the Scrittura parser class for Structured Products is `com.ipicorp.scrittura.messages.GenericXMLStructureParser`.

If Structured Products are implemented in the Scrittura application, this parser should be used instead of the generic parser for standalone trades detailed in Standalone Trade Message Parsing, on the previous page.

Do not include both `GenericXMLParser` and `GenericXMLStructureParser`. The parser used for Structured Products can also handle standalone trades.

The `GenericXMLStructureParser` class should be added to the general Scrittura configuration in the `scrittura-config.xml` file, located under the config repository in Scrittura live folder. A new `<message-type>` tag is added under the root tag, `<scrittura-config>`, adding this class to the list of available parsers.

```
<scrittura-config ...>

...

<message-type name=”XMLStructuredMessage”

class=” com.ipicorp.scrittura.messages

.GenericXMLStructureParser” />

...

</scrittura-config>
```

Example

```
The following is an excerpt from the structured-product-config.xml

configuration file.

<structured-product-config>

...

<key-value>

<parser-component-xpath>

/FILE/ITEM

</parser-component-xpath>

<key-values>

...
```

```
<structured-product-config>
```

Assume that the following incoming message is received:

```
<FILE>
<ITEM>
<messageID>ID001</messageID>
<tradeDate>2013-05-31</tradeDate>
</ITEM>
<ITEM>
<messageID>ID002</messageID>
<tradeDate>2014-12-03</tradeDate>
</ITEM>
</FILE>
```

With such a configuration, the incoming message is divided into the following XML messages, subsequently reinserted into the workflow to be parsed individually:

Message 1

```
<ITEM>
<messageID>ID001</messageID>
<tradeDate>2013-05-31</tradeDate>
</ITEM>
```

Message 2

```
<ITEM>
<messageID>ID002</messageID>
<tradeDate>2014-12-03</tradeDate>
</ITEM>
```

Data Derivation

Data derivation is an important step in the Message Processing workflow. It ensures that data and all variables required by product creation or subsequent steps is populated as expected in the message ticket, regardless as to how the information is provided by upstream systems.

Scrittura can map a set of input data to another set of values by executing a set of translations on the product variables. It can also create new variables based on the input provided by the incoming message.

For example, an input may deliver currency values as "USD" or "BRL", which would need to be mapped to "United States Dollar" or "Brazilian Real" in a final confirmation or before handing off to another system.

Scrittura provides multiple ways to derive and map variables.

Using BLogic for Data Derivation

Scrittura business engine, BLogic, is the preferred and recommended way to perform data derivation. Rules are defined within Microsoft Excel spreadsheets and evaluated against the PI at runtime.

BLogic use is not limited to the Message Processing workflow and can be used in any other workflow if required. For more information, see BLogic Business Engine, on page 203.

Using BeanShell Scripts for Data Derivation

A workflow can include a BSH activity (DataDerivation) that calls a script (for example, DataDerivation.bsh).

Within that script, additional scripts may be invoked which reflect a series of translations.

To invoke additional BeanShell scripts from inside a script, use the

```
pi.runBSHLogic command:
```

```
pi.runBSHLogic("PartyBSearching.bsh");
```

```
pi.runBSHLogic("TranslateCurrencyCodes.bsh"); pi.runBSHLogic("TwoLetterCodes.bsh");
```

The following is an example of setting Product Instance variables in a BeanShell script:

```
if(foo.equals("bar")) { Currency = "United States Dollar"; }
```

Using Classtools for Data Derivation

Classtools can also be used for data derivations and should be preferred to Bean Shell scripts performance-wise. In this case, the data derivation logic is coded in a Java class, passed as parameter to the classtool.

Unlike BLogic or BeanShell scripts, any change to the classtool logic requires rebuilding and redeploying the application.

Message Sequencer

The Message Sequencer Module lets the user to define Sequenced Sections within the workflow.

A Sequenced Section is a section of the workflow in which messages belonging to the same business entity are guaranteed to be processed in order of their arrival. It also ensures that only one work item related to a specific business entity is processed in the Sequenced Section at the same time.

A work item belongs to a business entity based on specific criteria as defined by the implementation. Multiple work items related to one business entity are processed by the workflow in a certain sequence in order to maintain the related entity's data consistency.

Note the following:

- A business entity can be a trade or a structured trade.
  - A trade is all work items with the same reference.
  - A structured trade is all work items linked by the same reference.
- Trade amendment messages for the same trade must be processed in the Message Processing workflow in their order of arrival, so that an older amendment does not override a more recent one.

For details, refer to the corresponding Javadoc for package and individual class level documentation.

Data Model Setup

The following tables are required for the Message Sequencer module.

- `itemseq_state`. Maintains state information of the last processed message for a sequenced item. New incoming messages for the same sequenced item are allowed through if no sequenced messages exist for that item, or if all prior messages have been completely processed. If a message is currently being processed in the Sequenced Section, new incoming messages are routed to a queue activity in the workflow.
- `itemseq_queue`. Maintains sequence information on queued items in the workflow. When a message exits the Sequenced Section, this table determines which message is processed next.

The creation scripts are provided with the Scrittura distribution for the supported database types: Oracle, Sybase, and SQL server. The creation scripts are located under the sql directory of the Scrittura distribution.

Workflow Setup for Message Sequencer

A Sequenced Section can be set up in the workflow using two classtools:

- SequenceBeginClassTool
- SequenceEndClassTool

Each of these classtools uses the following properties.

Package	Extended Attributes
`com.ipicorp.scrittura.sequencer`	This classtool uses two mandatory attributes: ◦ `sequencedSectionId` - the sequence identifier ◦ `queueActivityId` - the manual queue for held items

Items must enter the Sequenced Section through the `SequenceBeginClassTool` and must exit the section through the `SequenceEndClassTool`. A manual queue, such as `Message_Sequence_Queue`, must be set up with a transition leading to it from the Sequence Begin section, with routing based on the Boolean property scrHoldSequencedMessage set to true.

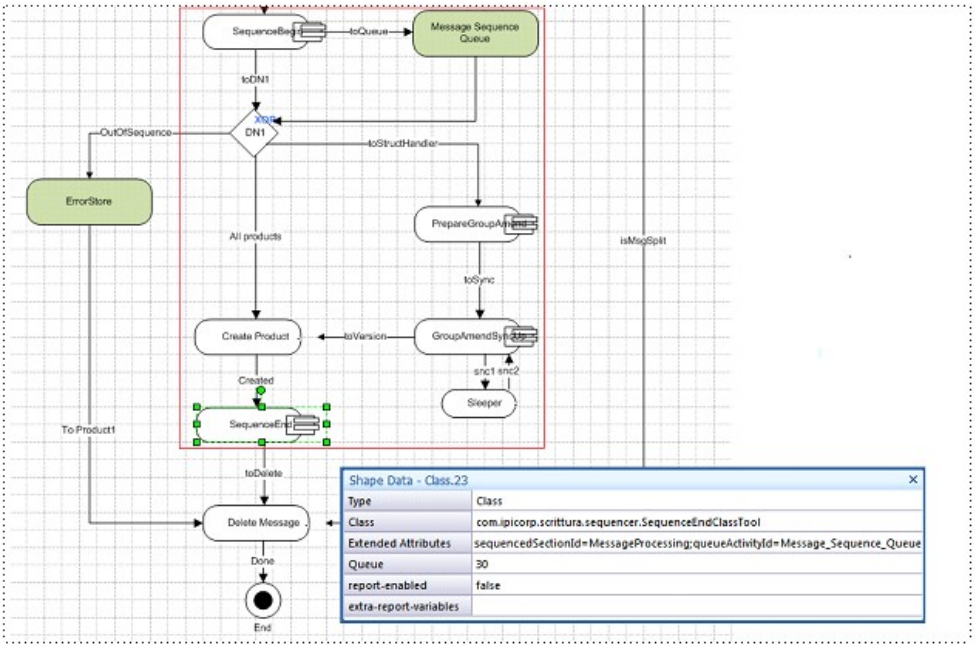
If an item is deemed out of sequence, a workflow transition using the Boolean property `scrOutOfSequence` can be used to prevent the item from entering the Sequenced Section. The item is then routed out of the regular workflow.

Multiple or nested Sequenced Sections can be set up in a workflow. The particular section is identified by the extended attribute `sequencedSectionId`, which must be set to the same value for a matched pair of sequence begin and sequence end steps. This `sequencedSectionId` name must match the section name in the `sequencer.properties` file. See Message Sequence Configuration, below.

The sequence end class must also be provided with the additional extended attribute `queueActivityId`, which identifies the manual queue which will store the queued items for that Sequenced Section.

Example

The following figure provides an example of a Message Processing workflow that sequences the handling of incoming messages. The Message Processing workflow determines whether the incoming message is a new or amended trade, applies different processing rules, and creates a new PI or amends an existing one.



Message Sequence Configuration

The Message Sequencer configuration is performed in the `sequencer.properties` file, located under the config directory of the Scrittura live folder.

The following properties can be specified.

Property	Description
section.{seq name}.itemTypeDetector.class	Class name of the item type detector for the Sequenced Section name. The section name, specified by {seq name} in

	the property name, is the same as the extended attribute `sequencedSectionId` specified for sequence begin and sequence end classtools.
item.{item type}.type	Name of the item type returned by the item type detector, whose value should also be {item type}. All other properties that start with item.{item type} are associated with this item type.
item.{item type}.handler.class	Class name of the handler for the item type.
item.{item type}.singleElementQueue	Determines behavior of the sequencer when it queues items because another item is already being processed in the `Sequenced Section`. Generally, when an item leaves the Sequenced Section, the sequencer picks another item with the lowest sequence number from the queue and sends it to the Sequenced Section. If this property is present and is set to true, the sequencer picks the item with the greatest sequence number. The sequencer then marks all other queued items as `OutOfSequence` and sends them away from the queue. This property is useful for improving system performance by filtering out unnecessary items if items with greater sequence numbers override the effects of items with lower sequence numbers.
item.{item type}.sequenceIdField	Field from the incoming message that holds the `Sequence ID`. Typically for standalone trades, this is the `CommonReferenceID`.
item.{item type}.sequenceNumField	Field from the incoming message that holds the Sequence Number.

Example

The following excerpt is an example from the `sequencer.properties` configuration file:

```
section.MessageProcessing.itemTypeDetector.class
=com.ipicorp.scrittura.sequencer.MessageItemTypeDetector item.trade.type        =trade
item.trade.handler.class
=com.ipicorp.scrittura.sequencer.TradeMessageTypeHandler
item.trade.singleElementQueue =true
item.trade.sequenceIdField      =idTrade item.trade.sequenceNumField    =sequenceNum
```

In this example, the name of the `Sequenced Section` is `MessageProcessing`. Upon receiving a message, when the Item Type Detector class (in this example, `MessageItemTypeDetector`) returns the value trade, the handler class `TradeMessageTypeHandler` provides the actual Sequence ID.

Interface Implementation

This section explains how to implement the interfaces for Type Detector classes and Item Type Handler classes.

Type Detector Classes and Item Type Handler Classes require different interfaces.

Type Detector Classes

Type detector classes must implement the following interface. This is the interface for the object that looks at a work item and detects the item type so that an appropriate item type handler can be selected.

`com.ipicorp.scrittura.sequencer.ItemTypeDetector`

The following parameters apply to Item Type Detector.

Class	`MessageProcessingTypeDetector`
Package	`com.ipicorp.scrittura.sequencer.impl`
Description	Concrete type detector class that returns a type of SimpleTrade

Item Type Handler Classes

Item type handler classes must implement the following interface. This is the interface for specific item type sequencing handlers. Handlers are responsible for determining the sequence ID, sequence number, and other properties.

`com.ipicorp.scrittura.sequencer.ItemTypeHandler`

The following parameters apply to Item Type Handler.

Class	`SimpleTradeHandler`
Package	`com.ipicorp.scrittura.sequencer.impl`
Description	Concrete item type handler class that returns the value of `CommonReferenceID` in the Message Ticket as `Sequence ID`. To be valid, `CommonReferenceID` must be populated prior to reaching the `Sequenced Section`.

Common Use Case

The most common use case is standard in the distribution: handling single trades as a business entity.

In this use case, amendments and events for a specific trade are processed by the system exactly in their incoming order. Subsequent events wait in the Message Sequence Queue as long as the current event processing within the Sequenced Section is not complete.

User Interface for Message Sequencer

Two JSP screens are provided for display purposes to view the content of the Message Sequence Queue. These screens can be integrated into the Scrittura user interface if required. No actions are available from the JSP screens.

Message Sequence Queue

JSP Name	`sequencerQueueView.jsp`
Description	This JSP displays the sequences for the Sequenced Sections. A link to the Sequence Items List is provided for each sequence.

Sequence Item List

JSP Name	`sequenceItemsView.jsp`
Description	This JSP displays the different sequences for the selected sequence.

Chapter 12: Outbound Workflows

The topics in this section describe important steps and components involved in outbound trade processing workflows, their configuration, and interfaces.

This section contains the following topics:

Document Generation Overview

Document generation is core feature of the Scrittura platform, which includes draft document generation (for example, as a Word document), subsequently converted to its final form in PDF format before being sent out to the counterparty.

Document generation is based on templates, which are evaluated during this process against the data of the trade for which the document is generated. Usual features provided by templates include, but are not limited to, trade data insertion, evaluation of conditional logic, inclusion of subtemplates, and barcode insertion.

Scrittura supports different types of templates for document generation purposes.

- **HTML templates.** HTML allows templates to be designed as HTML pages, enhanced by a set of instructions provided by Scrittura.
- **JSP templates.** JSP allows templates to be designed as JSP pages using the Scrittura core tag library.
- **Microsoft WordML templates**. WordML templates combine the usual Microsoft Word interface and its enhanced capabilities with Scrittura custom tags.

  > **NOTE:** Although WordML capabilities are supported by the Scrittura platform, WordML has been removed from Microsoft Word since its late 2007 version, making this module only usable with Microsoft Word 2003 and early versions of 2007.

- **Microsoft DOCX templates.** DOCX templates are the recommended solution. They combine the user-friendly Microsoft Word interface introduced by Word 2007, powerful Microsoft Word document edition capabilities with the extensive set of instructions provided by Scrittura.

While all of these document generation modules are supported by Scrittura, the use of DOCX templates is the recommended document generation solution. The Document Generation Suite provides the full suite of tools and servers to handle DOCX templates.

The remainder of this section summarizes how DOCX document generation is integrated into the Scrittura workflow. Full details, including how to design templates, setting up and deploying the necessary servers, can be found in the Document Generation Suite documentation.

> **NOTE:** For document generation based on HTML, JSP or WordML templates, see the previous versions of the documentation.

Draft Document Generation

Draft document generation consists of creating a Word DOCX version of the document that will be subsequently reviewed before the final PDF version is generated (or directly converted into the final PDF would the trade be STP).

The following classtool is provided for easy integration into the workflow.

Name	Word2007GenDoc
Class	`com.ipicorp.scrittura.docgen.word07.Word2007GenDoc`
Extended Attributes	◦ `templateName`. Name of the template to be used by DocGen without the DOCX extension. ◦ `documentTitle`. Document title as it will appear in the DocManager. ◦ `makeDefault`. Boolean indicator that, when set to true, the generated document is the default document. ◦ `performLargeDocumentOptimisation`: Boolean indicator that, when set to true, activates an optimization for large documents (see Lengthy Document Generation Tasks, on the next page).
Output	DOCX confirmation, saved against the current Product Instance.

Alternatively, PI variables can be used rather than extended attributes; those are defined in docgen-config.xml. Extended attributes take precedence over those PI variables when specified.

PDF Document Generation

Documents generated in PDF format can be simpler and safer for distribution to counterparts.

The following classtool is provided for integration into the Scrittura workflow in order to convert the draft DOCX document into PDF.

Name	`GenPdfDocClassTool`
Class	`com.ipicorp.scrittura.docgen.word07.GenPdfDocClassTool`
Extended Attributes	◦ `sourceTitle`. Name of the .DOCX document that is stored against the Product Instance. This value must match the documentTitle attribute of the Word2007GenDoc classtool. ◦ `targetTitle`. Name of the PDF document as it is stored in the

	Product Instance. ◦ `makeDefault`. Boolean indicator that, when set to true, the resulting document is the default document. ◦ `refresh`. Boolean indicator that, when set to true, the draft document is refreshed with the latest variable values prior to generating the PDF document. ◦ `acceptRevisions`. Boolean indicator (defaults to false). When set to true, all changes are accepted in the document prior to PDF generation. ◦ `saveValidatedDOCX`. Boolean indicator (defaults to false) that works in conjunction with the acceptRevisions flag. When set to true, the resulting DOCX, with all changes validated, is also saved against the PI. ◦ `performLargeDocumentOptimisation`: Boolean indicator that, when set to true, activates an optimization for large documents (seeLengthy Document Generation Tasks, below)
Output	PDF confirmation, saved against the current Product Instance. if `acceptRevisions` is set to true, the DOCX document with accepted revisions is also returned.

Alternatively, PI variables can be used rather than extended attributes; those are defined in `docgen-config.xml`. Extended attributes take precedence over those PI variables when specified.

Lengthy Document Generation Tasks

DOCX document generation is generally an immediate task showing good performance in an appropriately scaled system.

In the specific cases where large documents are generated, the process may require a far longer time than usual to complete. Although this would not be a problem in a standard workflow, some database deadlocks would be frequently observed when parallel document generation steps are performed simultaneously against the same Product Instance.

To circumvent this limitation, the performLargeDocumentOptimisation extended attribute can be specified in the Word2007GenDoc and GenPdfDocClassTool classtools. By setting its value to true, the lengthy document generations or PDF conversions are isolated in their own transaction, allowing multiple similar tasks to take place at the same time.

Bulk and Document Signatures

This section describes the signature process in Scrittura and only applies to signing documents generated with Document Generation Suite. The signature process for documents generated by Scrittura legacy solutions (HTML, JSP, WordML) is supported and remains unchanged (see the previous Scrittura Administration guides).

You can add a signature to a document either automatically in the workflow (using a set signature graphic), manually for single documents, or manually in bulk, based on a particular user's approval (using an individual user's graphic).

Bulk and Document Signature Configuration

This section details how to configure the signing process in Scrittura.

signatures.xml

The first information to specify in signatures.xml is the entity type used by DocManager to store signatures. This is specified by the entity-type attribute of the main node, signatures.

```
<signatures entity-type=”Signatures”>
```

Each user with signatory rights needs an entry in signatures.xml. For example:

```
<signatures entity-type=”Signatures”>

<signatory id="jack"

image="jack.gif" name="Jack Hilltumbler" title="Senior Manager"/>

<signatory id="jill"

image="jill.gif" name="Jill Hilltumbler" title="Director"/>

</signatures>
```

These users should all share a particular role. For example, the users "jack" and "jill" in the example defined above could be members of a role called "Signers," defined in the application server.

DocManager Signature Storage

A folder named Signatures of entity type Signatures should be created in DocManager, and all signatures should be stored in that folder. It is possible to use the SetConfig process to load the actual signatures into the folder.

For example, the graphics mentioned in signatures.xml (such as,jack.gif, jill.gif) should be saved in the Signatures folder.

Product Definition Setup

The signature PI variables must be defined in the trades’ Product Definition (typically shared in commonvars.xml), as arrays whose dimension is the number of requested signatures.

One or more signatures can be defined, as there may be requirements for more than a single signature. In the example below, two signatures are defined.

```
<VariableDefinition valueType="String"

audit-type="signature" subscript1ListLimit="2">

<internalName>SignatureDocx</internalName>

<visibleName>Signature</visibleName>
```

```
<defaultValue/>
</VariableDefinition>
<VariableDefinition valueType="String"
audit-type="signature" subscript1ListLimit="2">
<internalName>SignatureImageDocx</internalName>
<visibleName>Signature Image</visibleName>
<defaultValue/>
</VariableDefinition>
<VariableDefinition valueType="String"
audit-type="signature" subscript1ListLimit="2">
<internalName>SignatureNameDocx</internalName>
<visibleName>Signature Name</visibleName>
<defaultValue/>
</VariableDefinition>
<VariableDefinition valueType="String"
audit-type="signature" subscript1ListLimit="2">
<internalName>SignatureTitleDocx</internalName>
<visibleName>Signature Title</visibleName>
<defaultValue/>
</VariableDefinition>
```

> **NOTE:** Arrays of signature variables to use with the DGS require numeric indexes. A “Docx” suffix has been added to the variable names in order to avoid confusion with the legacy set of variables, would those be used elsewhere.

Next, the different levels of signatures must be configured in the different Product Definitions to specify whether a signature is either "user" based or "group" based. For example:

```
<signature level="1" type="user" role="Signers"/>
<signature level="2" type="group" role="Signers"/>
```

The type defined "group" vs. "user") also has an impact on the "id" attribute and its values in the signatures.xml file. If "user" is defined, userIDs (such as, "jack") in the signature.xml attribute is populated as opposed to the value "group", where an actual group (such as, Signers) is populated in that field.

Manual Signature

In the scrittura-config.xml file, you can define different views for a queue. “Bulk Review” displays the list of trades belonging to a queue and allows bulk actions. “Review” displays view economic details

and documents for a single trade. A manual signature can be applied to a single trade through the “Review” view or a series of trades through the “Bulk Review” view.

This is accomplished through a Next Action panel (`nextAction.jsp` or `nextActionWithAnn.jsp`) in the signature queues (bulk and review views). The list of next steps for this panel in the signature queue, defined in general-ui-`config.xml`, should contain the signature step specified by adding `SignatureProcess` as Process Handler for that step.

Example

```
<next-step value="signer_1_"display="Sign & Fwd"
singleTradeHandler=”com.iwov.….SingatureProcess”/>
```

The following are the characteristics of the SignatureProcess handler.

Class	SignatureProcess
Package	`com.iwov.gcm.scrittura.web.queue.processhandlers`
Description	Applies the signature variables to the selected PI. The signature level is determined from the value attribute, which must end by *X* or *[X]* where *X* is a number equal to the signature level (such as, 1, 2).

NOTE: A user-based signature for the same user can only be applied once to a document.

Once applied (option selected and trade forwarded), the signature process is handled by the Scrittura controller and signature variables are set accordingly in the PIs.

The trade is then forwarded to its next workflow destination. Consequently, the workflow should have a transition from this signature queue to its next destination based on the queue route variable holding the value that triggered the signature process (“signer_1_” in the example).

Automatic Signature

A workflow may be configured to automatically add signature images before dispatch, without human review.

The following classtool is provided to automatically add signatures.

Classtool	`ApplyDocxSignature`
Package	`com.ipicorp.scrittura.signature`
Attributes	List of “signer_X” attributes where X is a number (such as, 1, 2, and so on) whose values are signatory IDs.

This classtool applies signature variables to the PI as specified in the attribute list. It is possible to use the content of another PI variable to specify the value of the signer_X attributes, in which case, the name of that PI variable should be within brackets {…}.

Example: Use the content of PI variable mySignature1 for the first signature.

signer_1={mySignature1}

Remove Signatures

You can set up your workflow to unset signature variables in cases where the document is being amended and will require re-approval.

The following classtool is provided to remove signatures.

Classtool	`RemoveDocxSignature`
Package	`com.ipicorp.scrittura.signature`
Attributes	none

`RemoveDocxSignature` resets the different signature variables in the PI to empty values.

Apply Signatures to the Document

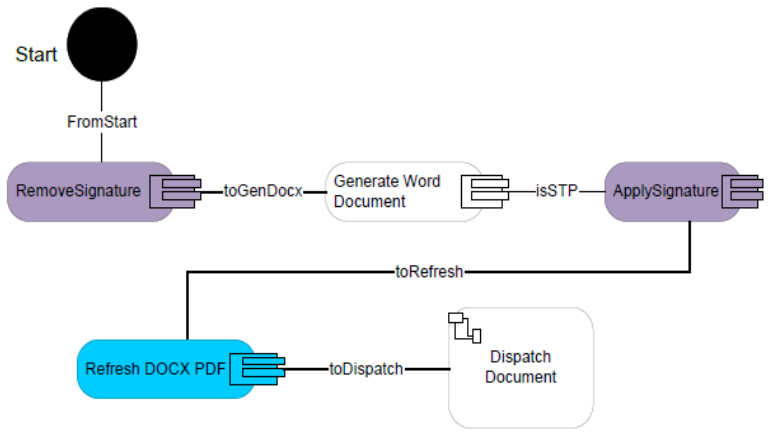
Once the signatures have been set in the Product Instance variables (automatically or manually), the signatures must be applied to the final document prior to the document being sent out to the counterparty. Rather than signing the draft document and generating the PDF from that signed draft, it is recommended that you only apply the signature to the PDF version of the document in order to generate the final (signed) document.

For this purpose, the workflow should include a step to refresh the PDF with the signatures, or create the PDF if none has been generated. This is done using the PDF Conversion classtool, `GenPdfDocClassTool`, with the extended attribute refresh set to `true`.

> **TIP:** If change tracking is enabled on the DOCX document, revisions should be accepted in order to generate a clean final PDF. This is done by setting the acceptRevisions attribute to true in the PDF Conversion classtool.

Automatic Signature Workflow Example

The following is an example of an automatic signature workflow.



The first step removes the signature, in case some were applied previously (for example, if the trade is amended). The draft DOCX is then generated (“Generate Word Document"), signatures are applied to the PI variables (“Apply Signature”), and the PDF is created (“Refresh DOCX PDF”). The document is now signed and ready to be sent.

Electronic Messaging

In addition to paper confirmation, electronic messaging is also supported by Scrittura for the following systems:

- DTCC DerivSERV
- ICE eConfirm
- SWIFT

For more information about Scrittura's electronic capabilities, see Electronic Messaging, on page 344.

Hand Off to a Fax Server

The CopyDocUtil utility class (located in the com.ipicorp.scrittura.util package) allows hand off of documents to a fax server.

Most popular fax servers can pull recipient information from either the first few lines of a file to be faxed or from a supplemental control file that points to the document to be faxed.

In case of an ASCII-formatted document (such as HTML), variable values (such as faxNumber, counterparty, and so forth) can be added at the top of the HTML document itself. The following sample code can be used in a beanshell script:

//When using a VAR defined without Arrays in product definition

```
String faxNumber = (String)(pi.getValue("faxNumber"));
```

//When using a VAR defined with Arrays in product definition

```
//String faxNumber = (String)(pi.getValue("faxNumber[A]")[0]);
```

// prepend pi data on top of html document

```
documentTitle = "Confirmation";
```

```
destinationFile = "c:\\opt\\scrittura3\\faxout\\fax_"
```

```
+ pi.getCommonRefId() + ".html"; prependControlString = "<TOFAXNUM:" + faxNumber +
">"; CopyDocUtil.performExtract( pi,
```

```
documentTitle, destinationFile, prependControlString);
```

Any graphic images in a generated HTML document are not stored directly in the document itself, but are referenced through a URL (such as `'http://hostname:port/images/logo.gif'`). These references must be resolvable by the fax server in order for the images to be properly transmitted.

In the case of a PDF document, any included images are available directly in the document. However, document content itself cannot be directly added as shown above. A control file must be

generated and dropped along with the PDF document. This control file can be configured to include fax server relevant data, available from the PI (such as faxNumber, counterparty, and so forth).

In the following sample BeanShell, note the use of 'controlFile' and 'controlText' to define the control file content and destination:

```
//When using a non-array variable

String faxNumber = (String)(pi.getValue("faxNumber"));

//When using an array variable

String faxNumber = (String)(pi.getValue("faxNumber[A]")[0]);

// define data of control file for pdf document documentTitle = "Confirmation";

destinationFile = "c:\\opt\\scrittura3\\faxout\\fax_"

+ pi.getCommonRefId() + ".pdf"; controlFile = "c:\\opt\\scrittura3\\faxout\\fax_"

+ pi.getCommonRefId() + ".ctrl"; controlText = "<TOFAXNUM:" + faxNumber + ">";
CopyDocUtil.copyDocAndGenerateControlFile( pi,

documentTitle, destinationFile, controlText, controlFile);
```

General Email Dispatch Capabilities

The preferred medium used to dispatch confirmations to counterparties from within Scrittura is generally fax, although it is required in some instances to send confirmations by e-mail.

Email messages are composed using the following fields and attributes.

Attribute	Description
TO	List of direct recipients of the email.
CC	List of recipients copied on the email.
BCC	List of recipients blind copied on the email.
FROM	Email address of the message originator.
SUBJECT	Subject of the email.
BODY	Content of the email.
ATTACHMENTS	List of files attached to the email.

Each of these fields is configurable in the Scrittura Email Dispatch module. Scrittura also provides simple template capabilities to generate the subject and body of those emails.

Email Dispatch Configuration

From within the workflow, enhanced email dispatch capabilities are available using the `DispatchEmail` classtool.

The `DispatchEmail` classtool does not have extended attributes but uses the values for different fields (such as TO or CC) from the PI variables. These variables are specified in the email dispatch configuration file, email- dispatch-config.xml, which is located under the /config directory. This file is used to configure email dispatch features and allows the user to define different PI variables and email templates. Email templates are placed under `/emailTemplates` in the Scrittura home directory.

The email-dispatch-config.xml file is comprised of the following nodes.

- **`<variables>`**. Defines the different PI variables to use for the different attributes (such as TO, CC, and so forth).
- **`<templates>`**. Defines the templates used to generate email the body and subject.

Example

```
<?xml version=”1.0” encoding=”UTF-8”?>
<email-dispatch-config>
<variables>
<to-recipient-list>scrEmailToList</to-recipient-list>
<cc-recipient-list>scrEmailCcList</cc-recipient-list>
<bcc-recipient-list>scrEmailBccList</bcc-recipient-list>
<from-recipient>scrEmailFromRecipient</from-recipient>
<attachment-list>scrEmailAttachmentList</attachment-list>
<attachment-visible-name-list> scrEmailAttachmentNameList
</attachment-visible-name-list>
<email-template-ref>scrEmailTemplate</email-template-ref>
</variables>
<templates>
<template name=”MainTemplate”>
<subject template="dispatch-header.txt" />
<body template=”dispatch.txt” />
</template>
<template name=”Chaser”>
<subject template="dispatch-header.txt" />
<body template=”chaser.txt” />
</template>
</templates>
</email-dispatch-config>
```

Product Instance Variables Configuration

Product Instance variables for email dispatch (`DispatchEmail` classtool) are defined under the `<variables>` tag.

`<variables>` has the following variables.

Variable	Description
to-recipient-list	Name of the variable that contains the list of TO recipients.
cc-recipient-list	Name of the variable that contains the list of CC recipients.
bcc-recipient-list	Name of the variable that contains the list of BCC recipients.
from-recipient	Name of the variable that contains the FROM field of the email.
attachment-list	Name of the variable that contains the list of attachments. Attachments are specified by their DocManager path.
attachment-visible-name-list	Name of the variable that contains the list of names under which the attachments will appear in the final email. If unspecified, attachment names will be taken from the list defined by the attachment-list variable.
email-template-ref	Name of the variable that contains the name of the email template to use, as specified in the **<templates>** node.

All list entries must be separated using commas, as in the following example:

`john.smith@bank.com,peter.john@bank.com,andrew.thomas@bank.com`

Example

Values for the different email attributes and fields are taken from the following PI variables:

- TO list: `scrEmailToList`
- CC list: `scrEmailCcList`
- BCC list: `scrEmailBccList`
- FROM field: `scrEmailFromRecipient`
- Attachment list: `scrEmailAttachmentList`
- Attachment visible name list: `scrEmailAttachmentNameList`
- Email template: `scrEmailTemplate`

  ```
  <variables>
  <to-recipient-list>scrEmailToList</to-recipient-list>
  <cc-recipient-list>scrEmailCcList</cc-recipient-list>
  ```

```
<bcc-recipient-list>scrEmailBccList</bcc-recipient-list>
<from-recipient>scrEmailFromRecipient</from-recipient>
<attachment-list>scrEmailAttachmentList</attachment-list>
<email-template-ref>scrEmailTemplate</email-template-ref>
</variables>
```

Email Templates Configuration

Templates used to generate the header and body of emails are specified as

`<template>` tags under the `<templates>` node. A `<template>` node has the following attribute.

Attribute	Required/ Optional	Description
name	Required	Name of the template, to which the variable defined under `<email-template-ref>` in the `<variables>` section refers.

A `<template>` node has the following child nodes.

Child Node	Required/ Optional	Description
subject	Required	This tag is empty and has one mandatory template attribute that specifies the subject email template to use. The path is relative to the `/emailTemplates` directory.
body	Required	This tag is empty and has one mandatory template attribute that specifies the body email template to use. The path is relative to the `/emailTemplates` directory.

Example

```
<templates>
<template name=”MainTemplate”>
<subject template=”dispatch-header.txt” />
<body template=”dispatch.txt” />
</template>
</templates>
```

Design Email Body Templates

Email body templates can be defined freely and are located under the `/emailTemplates` directory. These templates can be text or HTML files and can contain PI variables.

The following is the syntax to include a PI variable.

```
{field}
```

where field is a PI variable.

Email subjects, as defined in the email dispatch configuration file email- dispatch-config.xml, follow the same syntax.

Example template extract:

```
Dear Sir/Madam

I am writing to you with regard to transaction ref. {tradeID}.
```

produces the following output for a trade whose trade ID is 12345:

```
Dear Sir/Madam

I am writing to you with regard to transaction ref. 12345.
```

Integrate Email Dispatch with Scrittura

A classtool is provided in order to use email dispatch capabilities within Scrittura workflow.

Class name	`DispatchEmail`
Package	com.ipicorp.scrittura.util.email
Extended Attributes	none

This classtool does not have any extended attributes. Before a trade reaches this classtool, PI variables specified in the email dispatch configuration must be populated.

Chapter 13: Inbound Workflow

This section describes important steps and components involved in the inbound trade processing workflow, their configuration, and interfaces.

This section contains the following topics:

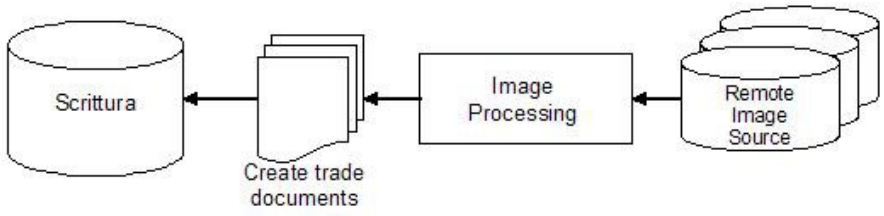
Image Processing Server

The Scrittura Inbound Image Processing Server (IPS) is a separate server that provides image processing and image transfer facilities to the main Scrittura system.

The system is designed by default to poll folders that contain images that have been created by some outside service or manual action. Upon receiving an image, the system processes it using any configured plug-ins that are available for that file format.

The results of this processing are used to create a description of the image containing any decoded information. This may be decoded barcodes, text recognized through Optical Character Recognition, or any other information discovered by the plug-in.

Inbound TIFF or PDF images may be split into multiple images if split barcodes are detected within the image. These are used as manual separators for the purpose of scanning batches of paper copies of documents for introduction into the workflow.



Once images have been processed, the image and the image descriptor file are transferred to a special drop box within Scrittura's domain and a dummy trade is sent to Scrittura. Scrittura is then able to act on these items as per the inbound workflow.

Image Process Server Configuration

The IPS configuration is configured using the following files, located under the `/config` directory of the IPS installation folder:

- config.xml
- log4j.cfg

- wrapper.conf

config.xml

The image processing server can be customized using the config.xml configuration file. . Its root node is `<config>`, under which the configuration is divided into the following distinct parts.

- image-processing, to configure image processing main properties
- log, to configure logs and alerts
- inbound, to configure inbound document processing
- outbound, to configure outbound document handling
- plugins, to configure plugins to be used by the IPS

Image Processing Configuration

The first step in configuring the Image Processing Server is to configure the image processing sub system, where all of the general properties will be set.

Example

```
<image-processing processing-folder=""

image-page-buffer="" processing-threads=""

document-conversion-format="PDF" perform-document-conversion="true" outbound-errored-docs="true">

<splitting classname="">

<param name="" value=""/>

....

</splitting>

<cover-page-detection>

<param name="" value=""/>

....

</cover-page-detection>

</image-processing>
```

The main node of the image processing sub system is image-processing. image-processing has two child nodes: splitting specifies document page splitting, and cover-page-detection specifies the detection of the cover page.

The `<image processing>` node has the following attributes.

Attribute	Required/ Optional	Description

processing-folder	Required	Defines the folder used for temporarily storing images retrieved from inbound folders.
image-page-buffer	Required	Defines the number of pages of images per document to buffer in memory. If sufficient memory is available, it is recommended that this value be set to a value equal to the expected unsplit document length.
document-conversion-format	Required	Defines the format of the document that should be sent on to Scrittura. Currently only PDF is supported. **TIP:** To convert from TIFF to PDF before processing, set the following attributes of the image-processing node in config.xml: ◦ document-conversion-format="PDF" ◦ perform-document-conversion="true" These modifications convert all documents to PDF
perform-document-conversion	Required	Specifies whether documents should be converted to the format specified by document- conversion-format.
processing-threads	Required	Specifies the number of threads for the Image Processing Server.
outbound-errored-doc	Optional	Specifies whether documents that fail to be processed (for whatever reason) should be sent to Scrittura. If the system is configured to do so, alerts are still dispatched for documents that fail to be processed successfully.

The `<splitting>` child node has a single attribute, classname, which specifies the custom class implementing the splitting algorithm to use. Required parameters for classname are defined as param child nodes.

The following are the splitting algorithms provided by the IPS.

- **Standard Split.** The Standard Split algorithm splits a document only if system barcodes matching the mandatory BarcodeMatchPattern parameter are detected on every page and unique distinct barcodes are present in consecutive chains.

  A standard split detection class is introduced to allow splitting of document where barcodes are detected on every page.

  This algorithm is implemented using the following classname attribute and param node:

classname	`com.scrittura.imaging.StandardSplitDetection`
param node	`BarcodeMatchPattern`, specified as a regular expression

- **Split based on Regular Expression.** This algorithm detects splits in a document based on the regular expression specified by the mandatory regex attribute. If a split is detected on the final page of the document, the final page will be lost.

  This algorithm is implemented using the following `classname` attribute and param node:

classname	`com.scrittura.imaging .SimpleRegExSplitDetection`
param node	`regex`, specified as a regular expression

The `<cover-page-detection>` child node has a single attribute, classname, which specifies the custom class implementing the appropriate cover page detection interface. Required classname parameters are specified as param child nodes.

The following standard implementation is natively provided by the IPS. This implementation detects the cover page and removes it if a barcode that matches the mandatory regex parameter is found on the first page of the document.

classname	`com.scrittura.imaging .SimpleRegExCoverPageDetection`
param node	`regex`, specified as a regular expression

> **TIP:** To use multithreaded capability, alter config.xml and use the processing-threads attribute of the image-processing node to specify the number of execution threads.

Alerts and Logs Configuration

Scrittura uses log4j for logging messages to a given destination. This may be the console, a text file, the Windows Event Log, or any other destination.

Alert messages are configured by defining the attributes of the <mail> child- node of the <log> node.

Example

```
<log>
<mail enabled="" smtp-server="" subject-prefix="" from="">
<contact recipient="" level=""/>
<contact recipient="" level=""/>
</mail>
</log>
```

The <mail> node has the following attributes.

Attribute	Required/Optional	Description
enabled	Required	Set to true to enable email reporting.
smtp-	Required	Specifies the SMTP server to use for sending mail.

server		
subject-prefix	Required	The prefix added to the subject line of any mail sent by the Scrittura system.
from	Required	Specifies the sender to be displayed for these internal messages.

The <mail> node is defined by one or more <contact> child nodes, one for each recipient to receive the email notification. <contact> has the following attributes.

Attribute	Required/Optional	Description
recipient	Required	Defines the email address of a single recipient for the alert message.
level	Required	Specifies the level of logging to send to the defined recipient. The level corresponds to the log4j log level. Possible values: `ERROR` or `FATAL`

Inbound Document Configuration

The inbound section specifies how incoming documents (such as those coming from the fax server) are handled by the IPS.

The `<inbound>` tag accepts one or more `<poller>` and `<blocker>` nodes, respectively to poll or wait for resources to be available.

`<poller>` and `<blocker>` each have a single attribute, classname, which defines a class that is called by the system periodically to poll some resource. Parameters to pass to this class are specified as `<param>` child nodes.

Example

```
<inbound>
<poller classname="">*
<param name="" value=""/>
....
</poller>
...
<blocker classname="">*
<param name="" value=""/>
....
</blocker>
</inbound>
```

The following examples are native implementations provided with Scrittura.

Example filesystem-based inbound source

The IPS comes with an implementation of simple drop box based communications. The class reads images from a local or remote file location (this can be a UNC address), polling that location for new files periodically. Example configuration:

```
<inbound>

<poller

classname="com.scrittura.imaging.DropBoxInboundSource">

<param name="path"

value="/opt/image-processing/test/outbound1"/>

<param name="poll-interval" value="3000"/>

<param name="sleep-interval" value="10000"/>

<param name="description" value="Local outbound folder"/>

</poller>

</inbound>
```

This example defines the following properties specific to the image processing sub system.

- **path.** Defines the folder to monitor.
- **poll-interval.** Defines the interval in ms to wait between polls of the folder.
- **sleep-interval.** Defines the interval, in milliseconds, to sleep in the case of repeated errors communicating with the source.
- **description.** Defines a description to give to the folder that will be used in error e-mails.

Example JMS inbound source

The IPS comes with an implementation of JMS Queue Consumer. The class reads images from a JMS Queue, blocking until an image is available.

```
<inbound>

<blocker classname="com.scrittura.imaging.JmsInboundSource">

<param name="java.naming.factory.initial" value=""/>

<param name="java.naming.provider.url" value=""/>

<param name="java.naming.security.principal" value=""/> <

<param name="java.naming.security.credentials" value=""/>

<param name="JmsConnectionFactory" value=""/>

<param name="JmsQueue" value=""/>

</blocker>

</inbound>
```

This example defines the following properties specific to the image processing sub system.

- **java.naming.factory.initial.** Defines the initial context factory, such as `weblogic.jndi.WLInitialContextFactory`.
- **java.naming.provider.url.** Defines the URL to use to communicate with the JMS server, such as `iiop://imageservice.scrittura.com:7101`.
- **java.naming.security.principal.** Defines the identity of the principal for authenticating the caller to the service. The format of the principal depends on the authentication scheme. If this property is unspecified, the behavior is determined by the service provider.
- **java.naming.security.credentials.** Defines the credentials of the principal for authenticating the caller to the service. The value of the property depends on the authentication scheme. For example, it could be a password with a hash (#), clear-text password, key, certificate, and so on. If this property is unspecified, the behavior is determined by the service provider.
- **JmsConnectionFactory.** Defines the JNDI name of the remote Connection Factory.
- **JmsQueue.** Defines the JNDI name of the remote JMS Queue.

Outbound Document Configuration

The outbound section specifies how output documents will be transferred from the IPS to Scrittura.

`<outbound>` defines the configuration of dropboxes to tie into Scrittura. The classname attribute of the `<outbound>` node defines the class that will be called by the system to communicate with Scrittura, whereas the `<param>` child nodes specify the parameters required for this class.

Example

```
<outbound classname="">
<param name="" value=""/>*
....
</outbound>
```

The following examples are native implementations provided with Scrittura.

Example dropbox based outbound sink

The IPS comes with an implementation of simple drop box based communications. The class delivers three files to Scrittura per image (unless that image is split) using dropboxes. The file formats of these classes are defined later in this chapter.

```
<outbound classname="com.scrittura.imaging.DropBoxOutboundSink">
<param name="trade-drop-box-path" value="/opt/scrittura/dropbox"/>
<param name="trade-drop-box-prefix" value="from_image_processing_"/>
<param name="trade-drop-box-suffix" value=".IN"/>
<param name="image-drop-box-path"
value="/opt/image-processing/test/image-drop-box"/>
<param name="image-drop-box-prefix" value="from_image_processing_"/>
<param name="image-drop-box-suffix" value="" />
```

```
</outbound>
```

This example defines the following properties specific to the image processing sub system.

- **trade-drop-box-path.** Defines the drop box used for trade tag files within Scrittura.
- **trade-drop-box-prefix.** Defines the string to be added as a prefix to any filename placed in the trade dropbox.
- **trade-drop-box-suffix.** Defines the string to be added as a suffix to any filename placed in the trade dropbox.
- **image-drop-box-path.** Defines the dropbox used for accompanying files within Scrittura.
- **image-drop-box-prefix.** Defines the string to be added as a prefix to any filename placed in the image dropbox.
- **image-drop-box-suffix.** Defines the string to be added as a suffix to any filename placed in the image dropbox.

Plugins Configuration

The `<plugins>` node defines the use of plug-ins to perform barcode detection. Zones can be defined to speed up processing of images when performing complex scanning operations, such as 2D barcodes.

```
<plugins>

<ocr classname="">

<param name="" value=""/>*

....

<zone top="" left="" bottom="" right=""/>*

....

</ocr>

</plugins>
```

`<plugins>` is defined by the `<ocr>` tag. The classname attribute of `<ocr>` defines the class name of the plugin, such as TaskbarPlugin. The `<param>` child nodes define a plugin-specific name/value pair (see the plugin documentation for details). The `<zone>` child node attributes (top, left, bottom, right), define a rectangular, in pixels, within the image that is to be scanned using the plugin. The variables `$WIDTH` and `$HEIGHT` can be used in conjunction with basic arithmetic.

Example <zone> child node configurations

- Scan the top 20% of any image:

  ```
  <zone top="0" left="0" bottom="0.2*$HEIGHT" right="0.2*$WIDTH"/>
  ```

- Scan the bottom 20% of any image:

  ```
  <zone top="0.8*$HEIGHT" left="0" bottom="$HEIGHT" right="$WIDTH"/>
  ```

- Scan the entire image:

```
<zone top="0" left="0" bottom="$HEIGHT" right="$WIDTH"/>
```

The IPS offers barcode detection capabilities, which are available by using a the Tasman Barcode plugin. For more information, see Tasman Barcode Detection Plug-in, below.

It is also possible to include other custom plug-ins; for more information see Image Processing Plug-ins, on page 324.

log4j.cfg

The system uses log4j for logging messages to a given destination. This may be the console, a text file, the Windows Event Log, or any other destination.

wrapper.conf

The Image Processing Service can use the wrapper from Tanuki Software in order to run as a service (or daemon) across multiple platforms.

The service is configured in the wrapper.conf file which is documented to explain the functionality that can be obtained.

Tasman Barcode Detection Plug-in

The Tasman barcode detection library provides a pure Java implementation for detecting most barcodes types including Code 128 and DataMatrix (see DataMatrix, on page 322). In order to use this plug-in an entry must be created in the `config.xml` file similar to the following example:

```
<plugins>
<ocr classname="TasbarPlugin">
<param name="code128" value="true"/>
<zone top="0" left="0" bottom="$HEIGHT" right="$WIDTH"/>
</ocr>
</plugins>
```

Additional logic allows for scanning only the first page of a doc or until the first barcode is encountered. These are set as configuration parameters to the Tasman barcode plug-in using `scanFirstPageOnly` and `scanUntilFirstBarcode` respectively.

This plug-in can render the PDF prior to scanning or extracts any images from the PDF through `extractPdfImages`. For more information, see `config.xml, on page 310` .

Barcode Zones

Zones define the areas of an image to search for barcodes. Defining zones can speed up processing, particularly when looking for complex symbologies such as DataMatrix bar codes.

It may be the case that the only location a barcode will be present is in the top right corner of the document, in which case a zone could be defined to process only this section:

```
<zone top="0" left="0.6*$WIDTH" bottom="0.2*$HEIGHT" right="$WIDTH"/>
```

In the case where the document is upside down, it is possible to define a mirroring zone covering the bottom left corner of the document:

```
<zone top="0.8*$HEIGHT" left="0" bottom="$HEIGHT" right="0.4*$WIDTH" />
```

Tasman Barcode Configuration

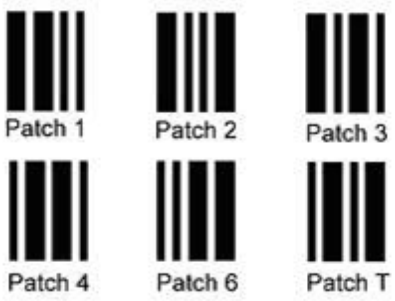
Barcode parameters can be set in the configuration file using the syntax:

```
<param name="variable" value="value"/>
```

The following are the available barcode parameters. All are optional, except for the `readxx` parameters (readEast, readWest, and so on), where at least one is required to be specified.

Parameter	Possible Values	Description
code128	true false (default)	Set to true to read Code 128 symbology barcodes.
code39	true false (default)	Set to true to read Code 39 symbology barcodes. Code 39 barcodes are similar to Code 32 barcodes with a different encodation. If both code39 and code32 are set to true, it is possible that one barcode will be returned for each specification.
codeDataMatrix	true false (default)	Set to true to read Data Matrix (ECC 2000) symbology barcodes.
readEast	true (default) false	Set to true to read the image with an Easterly scan (horizontally, left to right). Not relevant to searches for two-dimensional symbols, such as Data Matrix.
readWest	true (default) false	Set to true to read the image with a Westerly scan (horizontally, right to left). Set to true by default because images occasionally arrive upside down. Not relevant to searches for two-dimensional symbols, such as Data Matrix.
readNorth	true false (default)	Set to true to read the image with a Northerly scan (vertically, bottom to top). Not relevant to searches for two-dimensional symbols, such as Data Matrix.
readSouth	true false (default)	Set to true to read the image with an Southerly scan (vertically, top to bottom). Not relevant to searches for two-dimensional symbols, such as Data Matrix.

scanInterval	positive integer 5 (default)	Defines the scan interval. A value of 1 means scan every pixel row or column of the image; 2 means scan every second row or column, and so on. Increasing the value may give a faster reading of the image, but increases the probability of not detecting narrow or poorly formed barcodes. Decreasing the value of this field can have the opposite effect: slower reading of the image but better barcode detection. Faster reading with increasing values is not automatic. If the reader detects a structure in the image that is possibly a barcode, then the scan interval is temporarily decreased the scan interval in order to analyze the image. The value of this field is not usually relevant to searches for 2D symbols (such as Data Matrix), although for 'close-up' symbols the probability of successful reading is increased if scanInterval is not less than half the symbol's module (minimal square or 'blob') size.
code32	true false (default)	Set to true to read Code 32 symbology barcodes. Code 32 is also known as Base 32, Pharma 32/39, and Italian Pharmacode. It is not the same as Pharmacode-Laetus. Code 32 barcodes are similar to Code 39 barcodes with a different encodation. If both code32 and code39 are set to true, two barcodes may be returned (for the single original barcode), one for each specification.
code93	true false (default)	Set to true to read Code 93 symbology barcodes.
codeCodabar	true false (default)	Set to true to read Codabar symbology barcodes.
codeEAN13	true false (default)	Set to true to read EAN 13 symbology barcodes. EAN 13 barcodes are identical to UPC A barcodes with the same encoding except for an additional leading zero on EAN13 barcodes. If both codeEAN13 and codeUPCa are set to true, two barcodes may be returned (for the single original barcode), one for each symbology.
codeEAN8	true	Set to true to read EAN 8 symbology barcodes.

	false (default)	
codePatch	true false (default)	Set to true to read Patch Codes. The supported patch code types are shown in the figure below, where the read direction is from left to right. Symmetric patterns (2 and 4) also read correctly in the opposite direction, but other patterns read incorrectly (1 reads as 6, 3 as T, and vice versa).  A side effect of the relatively simple structure of patch codes is that they are liable to be reported when they do not actually exist in the image. An application that reads for patch codes should check that reported patch codes are of the anticipated type, orientation, size, and position within the image.
codeI2of5	true false (default)	Set to true to read Interleaved 2 of 5 symbology barcodes.
codePDF417	true false (default)	Set to true to read PDF 417 symbology barcodes.
codeTelepen	true false (default)	Set to true to read Telepen symbology barcodes.
code11	true false (default)	Set to true to read Code 11 symbology barcodes. Code 11 is also known as USD-8.
codeUPCa	true false (default)	Set to true to read UPC A symbology barcodes. UPC A barcodes are identical to EAN 13 barcodes with the same encoding except for an additional leading zero on EAN13 barcodes. If both codeEAN13 and codeUPCa are set to true, two barcodes may be returned (for the

		single original barcode), one for each symbology.
codeUPCe	true false (default)	Set to true to read UPC E symbology barcodes. UPC E encodings are a compressed form of UPC A. The Barcode.getString() method returns the full uncompressed UPC A encoding for this symbology.
plus2	true false (default)	Set to true to read 2-digit supplementals associated with EAN and UPC symbology barcodes.
plus5	true false (default)	Set to true to read 5-digit supplementals associated with EAN and UPC symbology barcodes.
separatorBarsToRead	positive integer 1 (default)	Defines the number of bars to attempt to read. Setting this value to the actual number of bars in the image can significantly improve performance when the image has more than two colors, or when searching for two-dimensional (such as, Data Matrix) symbols. The value of this field is not significant when searching two color images for one-dimensional or stacked symbols.
separatorDelta	positive integer 10 (default)	Specifies the pixel intensity step when scanning the image at values between separatorIntensity - separatorFlank and separatorIntensity + separatorFlank. NOTE Only used when the image has more than two colors.
separatorFlank	positive integer 0 (default)	Specifies the range of pixel intensity values above and below separatorIntensity with which to scan the image. NOTE Only used when the image has more than two colors.
separatorIntensity	1 through 255 128 (default)	Defines the intensity below which pixels are treated as black. The pixel intensity is calculated as the average of the red, green, and blue components.
code39NoGuard	true false (default)	Set to true to read Code 39 barcodes that are not delimited with the "*" guard character. Such barcodes do not meet the Code 30 specifiation and their use is not recommended. With this option set to true, the package can take a significantly longer time to read an image, 'proper' Code 39 barcodes may not be identified, the probability of reporting non-extant barcodes is significantly increased, and a checksum is not performed, even if specified.

code39NoGuardFFNX	true false (default)	Only significant if code39NoGuard is set to true. In this case set code39NoGuardFFNX to true to specify that the first and final characters in the barcodes do not include $ / + or %. This can reduce image read time and also reduce the number of non-extant barcodes reported.
smallQuietZone	true false (default)	Set to true to allow quiet zones of half the normal width. Quiet zones are the blank areas at the start and end of symbols. Their size depends on the barcode symbology, but is typically at least the width of a set of bars that encode a single character.
tinyQuietZone	true false (default)	Set to true to allow quiet zones of one third the normal width. If both smallQuietZone and tinyQuietZone are set to true, then quiet zones can be as small as one quarter of the normal width.

DataMatrix

DataMatrix is the two-dimensional barcode specification adopted by Scrittura for use in generated documents. The following illustration is an example of a DataMatrix barcode.

DataMatrix barcodes are preferable over other specifications for several reasons:

- They are very reliable. Even in the presence of faulty data they can be decoded successfully. This is very useful when faxing as image quality is poor.
- They can encode large amounts of data (up to 2000 characters, although this would not be practical for our purposes).
- There is no royalty payable for using the format.
- Scrittura supports generation of DataMatrix as standard.

> **NOTE:** The detection algorithms being very complex, scanning documents with 2D barcodes may take up to 4 times longer than documents with 1D barcodes.
>
> For these reasons it is advisable to be careful with the initial placement of 2D barcodes. If Scrittura just has to examine the top left corner of the document for the image then this will aid detection speed greatly.

Customization

The image processing system provides a framework on which large amounts of customization can be built; splitting and cover page detection algorithms, the communications protocols used, and the

classes that perform the actual image processing for barcode detection.

Custom Document Splitting

Documents may be split to separate one document from another that may have been concatenated during fax transmission or during a batch scan. The image processing module provides basic document splitting based around regular expressions, for configuration options of this splitting class see Image Processing Configuration, on page 310. In most instances, a more complex splitting implementation is required, an interface is provided for classes to extend for this purpose.

The Map `splitConfig` provides the developer with the name value pairs of parameters that are specified in the configuration file, enabling runtime parameters to be passed into the class.

Custom Cover Page Detection

Many documents may contain a fax cover page or a batch scanning cover page that can contains no useful information. A cover page detection class is provided for the removal of cover pages. This operation is performed after any document splitting has occurred on each split document. A simple regular expression basic cover page detection class is provided. This may not be sufficient and so the `CoverPageDetection` interface of the com.scrittura.imaging package is provided in order to implement custom classes.

This interface contains a single method, which returns true if the document has a cover page:

```
boolean hasCoverPage(Document doc, Map params);
```

The Map params attribute provides the developer with the name value pairs of parameters that are specified in the configuration file, enabling runtime parameters to be passed into the class.

Custom Inbound Communications

Two basic classes, `InboundDocumentSource` and `PolledInboundDocumentSource`, are provided to be extended to form the inbound communications module. It is also possible for multiple different inbound classes to be created and used simultaneously within the system.

Non Polling Based Communications

The class InboundDocumentSource provides a super class which should be extended in order to implement custom communications. The init method should perform initialization tasks. Custom parameters as defined in the configuration file are passed as name/value pairs in the params Map, the processingFolder is provided if it is required to store images to the local disk.

Finally the callback `InboundDocumentHandler` is provided to allow the class to inform the main system of received documents using the `receiveFile(File file)` or `receiveDocument(Document doc)` methods. The `run()` method should be implemented to perform the actions of interfacing with the document source.

Polling Based Communications

The class PolledInboundDocumentSource uses the same initialization method as InboundDocumentSource, taken from a common inbound interface. However rather than leaving the sole responsibility of interacting with the document source in the hands of the custom class, the `run`

`()` method is called periodically by the `InboundDocumentSource`. Documents are introduced to the system in the same manner as `InboundDocumentSource`.

Custom Outbound Communications

A single abstract class is provided in order to be extended to provide custom outbound communications. Three methods are provided that are called from the main system:

```
public abstract void init(Map params) throws InitialisationException;

public abstract void documentInbound(Document doc) throws CommunicationsException;

public abstract void documentOutbound(Document doc) throws CommunicationsException;
```

The `init` method is called upon system initialization; any parameters from the configuration file are passed as name/value pairs in the Map params.

The class is informed both when a document is introduced into the system (`documentInbound(...)`) and when it is finished being processed and all aspects can be sent on to Scrittura (`documentOutbound(...)`).

Image Processing Plug-ins

In order to interface with third party libraries for the purposes of adding barcode detection, plug-ins can be added to the system.

The image processing plug-in can use a zoom factor to increase the rendering resolution of PDFs to improve barcode functionality through the following startup options:

```
-Dips.zoom.factor=200.0f (200%)

or

-Dips.zoom.factor.first.page=200.0f (to zoom the first page)
```

TIFF Images and Browsers

For a user to see a TIFF image directly in the Scrittura application requires a browser plug-in, like Alternatiff. Scrittura does not include any browser plug- ins for TIFF images.

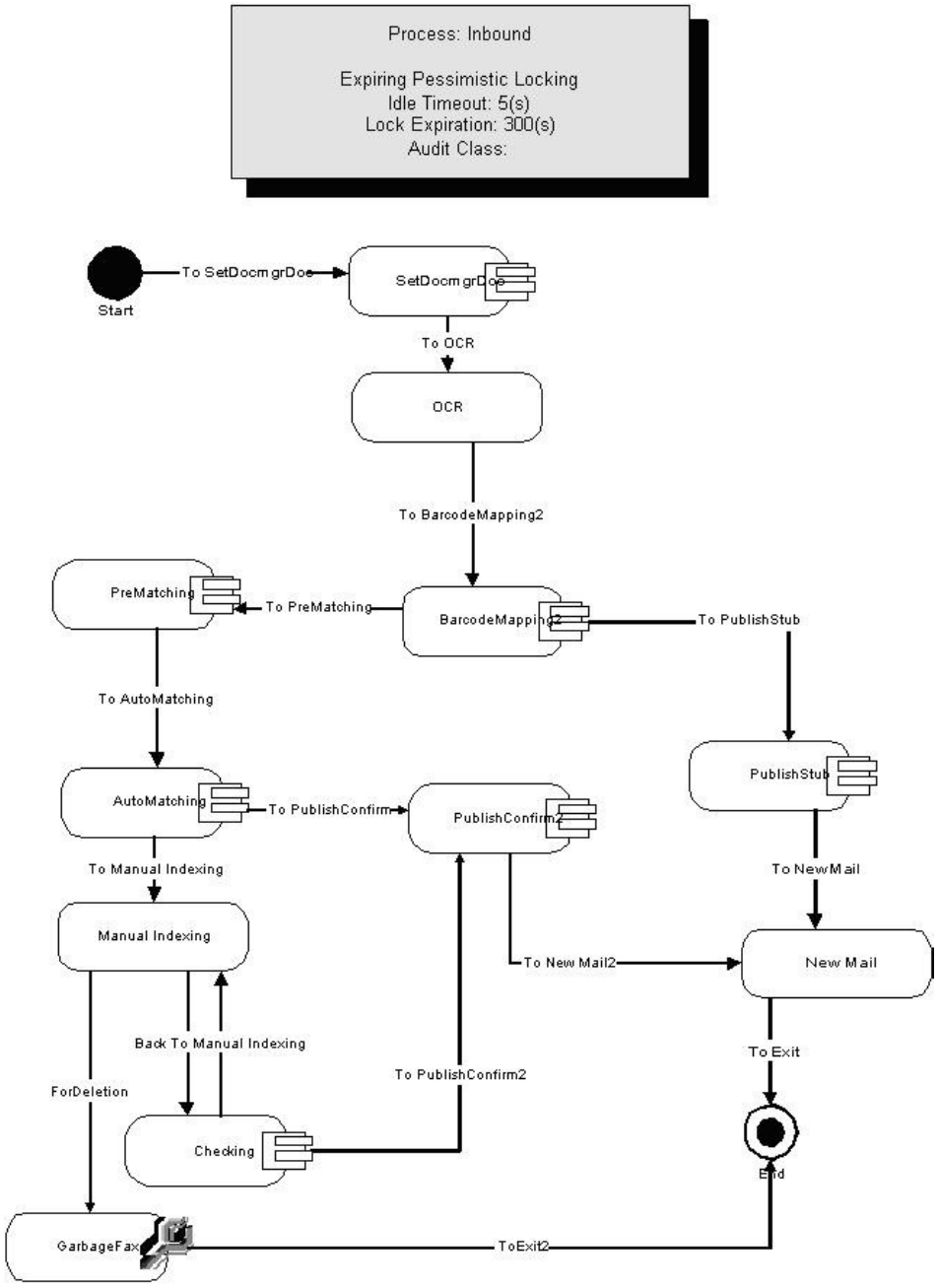
Some TIFF plug-ins may have different memory or browser requirements; be sure to read documentation on the plug-ins themselves to make sure they are compatible with your own network and configuration requirements.

Inbound Workflow

The following sample workflow serves to segregate:

- inbound images not containing recognizable barcodes
- inbound images containing recognizable barcodes which do not correspond to any previously filed document
- inbound images containing recognizable barcodes which do correspond to a previously filed document

The table that follows explains the function of each of the pictured class activities and the conditions on each of the "non-otherwise" transitions.



Workflow Activity	Class or Application Executed	Notes
SetDocMgrDoc	`com.scrittura.inbound.SaveInboundDocumen`	Extended attribute: `setStandardTiff=true`

	`t`	Performs housekeeping; associates the document (initially placed in the temporary ImportFolder) with its own Inbound Product Instance and moves it to its corresponding folder in DocManager. If the setStandardTiff extended attribute is true, the class will encode the TIFF as single strip and Group 4 (the standard requirements for a Scrittura TIFF).
OCR	*Manual Activity*	A manual activity not exposed to end users for interaction. The OCR daemon looks for waiting workitems in this activity, performs the OCR function and barcode recognition, and pushes the workitem along in the workflow.
Inbound TiffRotater	`com.scrittura.inbound.InboundTiffRotater`	Extended attribute: degrees=180 Activity that will rotate TIFFs. Degree of rotation depends on the value of the degrees extended attribute.
BarCode Mapping 2	`com.scrittura.inbound.BarcodeMapping2`	Looks at any recognized barcodes to see if they are in a recognized format. Currently, this class recognizes two: one for confirms - `"DC/IRS01/Confirmation" (DC/[CommonReferenceID]/Confi rmation)`, and one for DocManager 'stub' documents "ID+1234" `(ID+[DocManager Resource ID]).` The "DC" code ("Derivative Confirmation") is, for the moment, hard-coded into this routine and should be included in outbound barcodes. Transition to PublishStub only if `(BarcodeLength == 2) && (Barcode0.equals("ID"))`—in other words, if there are two strings in the barcode and the first is the string "ID". Documents without barcodes or other recognized barcodes go to the PreMatching activity.
PreMatching	`com.scrittura.inbound.Matching`	Adds the full-text of the TIFF to the workitem. Looks for items from the Matching Dictionary in the full-text of the TIFF, and adds those name/value pairs to the workitem. The Matching Dictionary is a flat file, matching.dict, located under the Scrittura \config folder.

AutoMatching	com.scrittura.inbound.Matching2	Attempts to auto-match this document to a trade already in the workflow, if a **CRID** is found in a recognized barcode format, or if a **CRID** was found in the full text of the document.
		
		`Transition to PublishConfirm if`
		
		`MatchStatus.equals("Barcode")`
		
		`or`

		`MatchStatus` `.equals("FullText")`
PublishConfirm	`com.scrittura.inbound.PublishConfirm2`	Extended attributes: ◦ **default = true** When set to true, the published inbound document will become the default document for the trade. ◦ **documentTitle=Signed** If this attribute is set, the published inbound document will have the name set in the "documentTitle" extended attribute. ◦ **createNewVersion=true** When set to true, the published inbound document will be versioned. ◦ **deleteInboundPI=true** When set to true, the inbound PI will be deleted after the class is done publishing. ◦ **deleteInboundDocs=true** When set to true, the inbound documents will be deleted after the class is done publishing. Attempts to publish the document to the trade folder to which it corresponds. Publishes the TIFF itself, the full-text of the TIFF as a .txt file, a file containing the name/value pairs of trade info, and a file containing an XML representation of those name/value pairs.
PublishStub	`com.scrittura.inbound.PublishStub`	Publishes the document to DocManager based on the DocManager Resource ID passed in by the barcode.

ManualIndexing	*Manual Activity*	Transition to GarbageFax only if
		`RealDoctype`
		`.equals("For Deletion"). This RealDoctype variable would be set by a user in the JSP view for this`
		manual activity.

NewMail	*Manual Activity*	A queue which serves as a collection point for all inbound documents so that the end user can see all inbound documents that have come into the system. These documents have either been processed automatically (barcode or CRID found in full-text) or have been manually processed.
Checking	`com.scrittura.inbound.SetPropBag`	The Matching manual activity provides a widget to give the user a spot to input a Common Reference ID. This Checking class makes sure that this value is valid and, if so,

		sets MatchStatus to Manual. Transition to `PublishConfirm` only if `MatchStatus.equals ("Manual")`.
Garbage Fax	*Workflow Application*: `Remove`	Deletes the document from DocManager. Subsequently the workitem is removed from the workflow.

Inbound Product Definition

Note that a Scrittura workflow configured as above requires a Product Definition to create an Inbound PI and attach the document coming from the fax server. This Inbound Product Definition contains specifications for the workflow variables referenced in this section.

Sample Implementations

This section presents examples of the Cover Page class, Document Splitting class, and DropBox Outbound Communications file.

Sample Cover Page Class

The following is a sample cover page detection class that illustrates the use of the interface.

```
package com.scrittura.imaging; import java.util.*;

public class SimpleRegExCoverPageDetection

implements CoverPageDetection

{

/*

If a barcode that matches the paramater "regex" is found on the first page of the
document then true is returned, else false is returned.

*/

public boolean hasCoverPage(Document doc, Map params)

{

if (doc.numberOfPages() < 1) return false; String regex = (String) params.get
("regex"); boolean match = false;

Page firstPage = doc.getPage(0);

if (firstPage == null) return false;

Iterator barcodeIter = firstPage.getBarcodes().iterator(); while
(barcodeIter.hasNext())

{

if (((String) barcodeIter.next()).matches(regex))

{
```

```
match = true;
}
}
return match;
}
}
```

Sample Document Splitting Class

The following is a sample document splitting class implementing the interface.

```
package com.scrittura.imaging; import java.util.*;
public class SimpleRegExSplitDetection implements SplitDetection
{
/*
Returns an array of integers (indexed from zero) representing the pages to be split around, i.e. for a 10 page document that it is decided that the document should be split as pages 1-5, 5-
10 then the array returned would be [0,4,5,9]. The split is inclusive of * both entries, i.e. 0-4 will return 5 pages.
Parameters:
doc - the document to be split.
splitConfig - a Map of name/value pairs from the configuration. Returns an even list of integers
*/
public List detectSplits(Document doc, Map splitConfig)
throws SplittingException
{
String regex = (String) splitConfig.get("regex"); if (regex == null)
{
throw new SplittingException( "No regex found.");
}
LinkedList list = new LinkedList();
// looking for pairs of split barcodes
for (int current = 0; current < doc.numberOfPages();)
{
```

```
boolean found = false;
// add current to list as first of a pair list.addLast(new Integer(current));
int next = current + 1;
for (; next < doc.numberOfPages(); next++)
{
//check through all barcodes on that page
LinkedList barcodes = doc.getPage(next).getBarcodes(); for (int i = 0; i <
barcodes.size() && !found; i++)
{
if (((String) barcodes.get(i)).matches(regex))
{
found = true;
}
}
if (found) break;
}
// if found or on last page then add entry if (found)
{
list.addLast(new Integer(next - 1)); current = next;
}
else
{
list.addLast(new Integer(doc.numberOfPages() - 1)); current = next;
}
}
// special case where whole document is selected if (list.size() == 2 &&
((Integer) list.getFirst()).intValue() == 0 && ((Integer) list.getLast()).intValue()
== doc.numberOfPages() - 1)
{
list.clear();
}
return list;
}
```

```
}
```

Drop Box Communications File Formats

Flat XML Tag File

Once an image has been successfully processed by the system a dummy tag file is created containing the filenames of the image and image descriptor files together with a unique common reference ID composed from the String InboundImage and a timestamp.

Example

```
<?xml version="1.0" encoding="UTF-8"?>

<product-instance>

<variable name="ProductDefID"> Inbound </variable>

<variable name="TradeType"> Inbound </variable>

<variable name="ImageFilename"> image-file-name.tif

</variable>

<variable name="ImageTextFilename"> image-file-name.tif.xml

</variable>

<variable name="CommonReferenceID"> InboundImage1083180395680

</variable>

</product-instance>
```

This file is transferred to the Scrittura trade drop box as defined in the configuration file.

XML Image Descriptor File

Once an image has been successfully processed by the system, an image descriptor file is created and sent along with the image and a trade tag file to a Scrittura drop box. This file contains an entry for each page detected in the image and any barcodes and text found. The following example shows an image that has been processed using a barcode detection library. A barcode was found on each page of the three pages of the TIFF image.

Example

```
<?xml version="1.0" encoding="UTF-8"?>

<document>

<page>

<barcode>LN-LN-SW-14471-50-36-1</barcode>

</page>

<page>

<barcode>LN-LN-SW-14471-50-36-1</barcode>

</page>
```

```
<page>
<barcode>LN-LN-SW-14471-50-36-1</barcode>
</page>
</document>
```

This file together with the original unaltered image is transferred to the image drop box as configured in the configuration file.

OCR Using Teleform and IDOL Image Server

Inbound trade confirmation within Scrittura now benefits from an enhanced level of automation by integrating with two other products:

- **Teleform.** Teleform is an Optical Character Recognition (OCR) application which is used to extract data from inbound trades to be matched with outbound trades.
- **IDOL Image Server.** IDOL Image Server is used to provide a confidence rating for signatures on inbound documents matching their signatures.

The Scrittura server then uses the economic data to match the inbound trade to a previously sent outbound trade and compares the confidence percentage of each signature to a minimum threshold to determine whether signatures on the inbound trade can be automatically approved or require routing for manual approval.

> **NOTE:** Teleform and IDOL Image Server are not bundled with the Scrittura platform. Check your license agreement for full information.

Scrittura OCR Solution

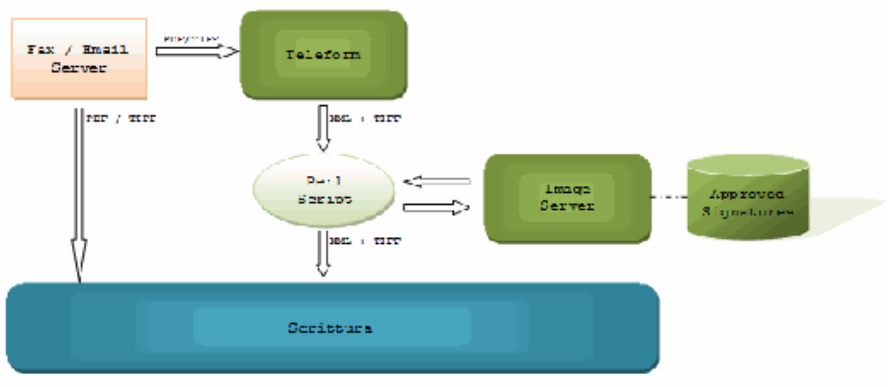
When an inbound trade is received, key economic fields can be extracted from the document along with its full text content. In addition, signatures can be extracted and matched against a base of approved signatures.

A confidence level is generated for each signature appearing in the inbound trade. Scrittura tests this confidence level against a configurable minimum confidence level threshold and simulates it in Scrittura queues and trade detail screens.

In addition, Scrittura native barcode recognition capabilities allow you to match inbound confirmations based on barcode detection.

Each of these features can be used separately or in combination as required.

The following illustration provides a general view of the different modules and how they interact.



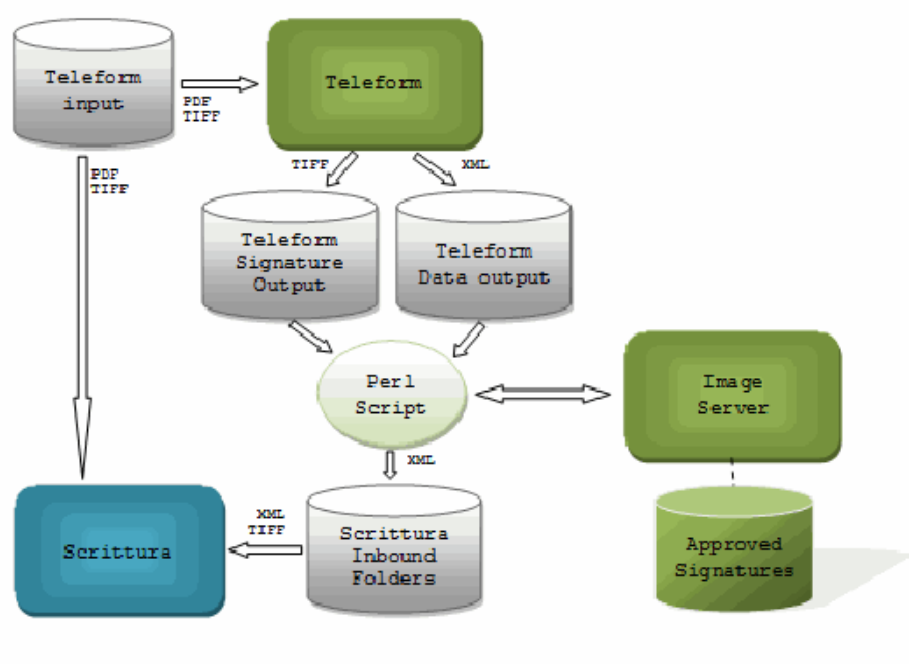
The modules communicate using file system repositories, or drop boxes.

Inbound documents are delivered from a fax machine to a drop box to be retrieved by the Teleform server. Teleform extracts signatures from the documents into TIFF files and uses OCR to extract economic data.

The economic data and signatures extracted by Teleform are subsequently processed by a Perl script. The Perl script formats the economic data for use by the Scrittura server, invokes IDOL Image Server to match the signatures against a set of approved signatures, and records the confidence percentage of the match in the document bound for the Scrittura server.

The Scrittura server then uses the economic data to match the inbound trade to a previously sent outbound trade and compares the confidence percentage of each signature to a minimum threshold to determine whether signatures on the inbound trade can be automatically approved or require routing for manual approval.

The following illustration details the different drop boxes involved in the interprocess communication.



Configure the OCR Components

Before beginning the configuration, you must install the Image Server and Teleform applications.

For details on the installation process, refer to IDOL Image Server and Teleform documentation. This section provides details on the configuration process for the following components.

- **DiSH.** DiSH is a component common to most products. It runs as a standalone service and manages licenses for the different components. Installing the DiSH is required to enable IDOL Image Server and Teleform services.

  Refer to general documentation for details on the installation and configuration process.
- **IDOL Image Server.** Signature recognition criteria is defined and configured in IDOL Image Server.
- **Teleform.** The Teleform OCR process is based on templates configured to detect key economic fields in inbound documents. Teleform output is an XML file that aggregates all information extracted from the document and signature information provided by IDOL Image Server.

Configure IDOL Image Server

The IDOL Image Server configuration process gives IDOL Image Server access to known signatures against which it will match signatures found in inbound documents.

To configure IDOL Image Server

1. In the `imageserver.cfg` file, locate the **Signature Template** setting under the **Detect Logo** heading.
2. Set the Signature Template setting to a folder that has the capacity to hold all the approved signatures.
3. Copy the signature images into the specified folder.
4. Restart I**DOL Image Server**.
5. Using a web browser on the **IDOL Image Server** box, query `http://localhost:18000/action=getstatus` to confirm that the number of signatures that you extracted into the folder appears in the `<logotemplates>` element of the displayed XML.

Configure Teleform

The Teleform configuration is performed using the Teleform Designer.

To fully configure Telefom, complete the following procedures.

1. Add templates to the **Teleform Designer**.
2. Designate the signatures for **IDOL Image Server** to match against those found on inbound documents.
3. Identify templates for incoming forms.

4. Configure the polling job.
5. Configure the drop box.
6. Configure the OCR engine settings.

Add Templates

Teleform uses templates to detect the structure of incoming documents and fields of interest.

To set up the templates to be used by Teleform during the recognition process

1. Open the **Teleform Designer**.
2. From the **File** menu, select **Templates**.

   The **Templates** dialog displays.
3. Right-click the right pane and select Import.

   The **Import** dialog displays.
4. Click Browse.

   The **Browse** for Folder dialog displays.
5. Navigate to the folder in which your templates are stored and click **OK**.

   Select the templates in the right pane and click Import.
6. From the **File** menu, select **Template Sets**.

   The **Template Sets** dialog displays.
7. Right-click the right pane and select **Add templates**.

   The **Select Templates** dialog displays.
8. Select the templates that you imported and click **OK**.

   The selected templates are added to Teleform.

Designate Signatures for Matching

This process designates the signatures for the IDOL Image Server to match against those found on inbound documents.

To designate the signatures for matching

1. Open the **Teleform Designer**.
2. From the **File** menu, select **Templates**.

   The Templates directory displays.
3. Select one of the templates that you imported and click **Open**.

   The template opens.

4. From the **Form** menu, click **Auto Export Setup**.
5. Select the appropriate line and click **Modify**.
6. From the **Format** list, select XML and then click **Save as**.
7. Select **Export** each record to a unique file in this directory and click **Browse**.
8. Select a folder for the XML OCR output and click **OK**.
9. Click **Done**.

   You return to the **Auto Export Setup** dialog.
10. Click **OK** twice to complete the process, then save your work.
11. Repeat steps 2 through 9 for each remaining template.

    The signatures uses for matching are designated.

Identify Templates for Incoming Forms

This process enables Teleform to recognize that an incoming form is associated with a particular template. That template is used to identify individual fields within the incoming form. Teleform recognizes keywords within the form which match it to the template. This configuration is stored in an IDJ file.

To enable Teleform to recognize an incoming form

1. Open the **Teleform Designer**.
2. From the **Utilities** menu, click **Identification Drop-In Setup**. The **ID Drop-In Setup** dialog displays.
3. Select **Full Page Keyword ID** and click **Import**.

   The **Open** dialog displays.
4. Select the IDJ file used to package the Full Page Keyword Identification rules and click **OK**.
5. The appropriate templates are identified for incoming forms.

Configure the Polling Job

For inbound TIFF or PDF documents, Teleform uses a drop box as the repository from which those documents are accessed for processing. Teleform scans the inbound drop box on a polling basis to retrieve incoming documents that have reached it.

To configure the polling job

1. Open the Teleform Designer.
2. From the **Utilities** menu, select **Job Configuration** and **click New**. The **Job Properties** dialog displays.

3. Type a name for your job and from the Error Handling list box, select **Automatically accept batch**.
4. From the **Form ID/Capture** tab, select **Forms Only** from the Expected types list.
5. From the **Identification** list, select **Use recognition set**.
6. Select the **Custom drop-in ID configuration** check box.
7. Click **OK**.

   The polling job is configured.

Configure the Inbound Drop Box

For inbound TIFF or PDF documents, Teleform uses a drop box as the repository from which those documents are accessed for processing.

To configure the inbound drop box for Teleform

1. Open the **Teleform Designer**.
2. From the **Utilities** menu, select **Connect Agent Setup** and click **Automated Batch Creation**.

   The **Automated Batch Creation** dialog displays.
3. Clear the Disable the creation of batches check box.
4. In the **Source Directory** text box, select a directory where the confirmation will be dropped.
5. In the **Pattern** text box, select *.*.
6. In the **Minimum number of image files per batch** field, select **1**.
7. In the Default batch job name drop-down menu, select the name of the polling job you created.
8. Click **OK**.

   The inbound drop box is configured.

Configure OCR Engine Settings

The final step in configuring Teleform is to set the character recognition properties.

Teleform is configured to prevent documents from being flagged for review by setting its **Confidence Threshold** value to **0** for character recognition. As the fields within inbound trades are matched by Scrittura, mismatches are reviewed in the Scrittura workflow rather than using Teleform to flag documents for review.

The **Batch Commit** setting is configured to commit the recognized fields automatically, allowing Scrittura to report any mismatches between outbound and inbound trades rather than waiting for a manual commitment by an operator.

Configuring the **Optimized for accuracy** setting configures Teleform to provide a more accurate recognition of characters rather than one that performs faster.

To configure the OCR engine settings

1. Open the **Teleform Designer**.
2. From the **Utilities** menu, click **Configuration**.
3. Click the **Reader** tab.

   Click the Recognition tab.

   From the **Character recognition** list, select **Confidence threshold**.

   From the **Character recognition** list, select 0.
4. From the **Reader** tab, click the **Local** tab.

   From the **Task** list, select **Batch commit**.

   Select the **Enable** check box.
5. From the **Reader** tab, click the **OCR Performance** tab.

   Select the Optimized for accuracy option.
6. Click **OK**.

   The OCR engine settings are configured.

Set Up Communication between IDOL Image Server and Scrittura

A Perl script, `SignatureID.pl`, is provided to format the economic data recognized by Teleform as an inbound message for the Scrittura server. This script also invokes the IDOL Image Server to provide a match confidence percentage for each signature in the inbound document against a repository of signatures from approved signers.

To run the script, you must install a Perl interpreter.

Custom parameters are included in the Perl script to match the custom environment. These parameters are defined under the `## config settings` comment line of the `SignatureID.pl` script.

Parameter	Description
$teleformHotFolder	Directory in which the inbound images are received.
$imageServerURI	URI of the IDOL Image Server.
$dirToOutput	Directory to which output containing signature confidence percentages is directed.
$dirToOutputImages	Directory to which output containing images of signatures and the original document from which the signatures are extracted is directed.
$dirSigLib	Directory in which the extracted signatures from inbound images reside.
@inputfiles	Pattern for recognizing input file names.
$imgExt	Extension used for files containing signature images.

When run, the script sends the formatted Teleform output to the Scrittura drop box specified by the `$dirToOutput` parameter.

Scrittura receives the corresponding inbound document from the path information specified in the message.

Signature Matching Configuration in Scrittura

The Signature Matching service is configured with the `inbound-config.xml` file, located in the Scrittura live configuration folder.

`inbound-config.xml` uses the structure described in this section.

Signature Matching

The `<signature-matching>` element contains the settings for the configuration of signature matching and has the following required attributes.

Attribute	Description
signatures-required	Defines the number of signatures required for matching
match-confidence-required	Specifies the confidence percentage required for each signature
partial-match-queue	Defines the name of the queue to which inbound messages are routed when not all of the values of variables in the inbound and outbound documents match
value-match-queue	Defines the name of the queue to which inbound messages are routed when the values of variables in the inbound and outbound documents match, but the minimum confidence percentage is not met for all signatures and/or the minimum number of signatures was not met
no-match-queue	Defines the name of the queue to which inbound messages are routed in case no match is found

Variable Mapping

This tag is used to specify the mapping between Teleform variables and internal Scrittura variables to capture Teleform output within Scrittura.

Variable mapping is configured under the `<signature-data-variables>` element. This element contains a collection of `<signature-data- variable>` elements and has no attributes. Any number of `<signature- data-variable>` tags can exist within the `<signature-data-variables>` element.

`<signature-data-variable>` has the following required attributes:

Attribute	Description

ocr-name	Name of the variable within the Teleform output
variable	Internal name of the variable within Scrittura

Example

Three Teleform variables are mapped against Scrittura variables.

Teleform Variable	Scrittura Variable
signature_1_confidence	`scrMatchPercentageSig1`
signature_2_confidence	`scrMatchPercentageSig2`
Buyer	`counterpartyName`

The corresponding XML configuration is as follows.

```
<signature-data-variables>
<signature-data-variable ocr-name="signature_1_confidence"
variable="scrMatchPercentageSig1"/>
<signature-data-variable ocr-name="signature_2_confidence"
variable="scrMatchPercentageSig2"/>
<signature-data-variable ocr-name="Buyer"
variable="counterpartyName"/>
<signature-data-variables>
```

Variable Matching

Variable matching is used to specify the name of an internal Scrittura variable that must match between the inbound and outbound documents for the inbound document to be approved automatically.

Variable matching is configured using the `<match-variables>` element. This element contains a collection of `<match-variable>` elements and has no attributes. Any number of `<match-variable>` tags can exist within the `<match-variables>` element.

`<match-variable>` has the following required attribute.

Attribute	Description
variable	Internal name of a variable that must match between the inbound and outbound documents for the inbound document to be approved automatically.

Runtime Inbound Process

This section details the steps to launch and run the inbound process with signature recognition. The runtime process requires the following components to be running:

- Teleform
- IDOL Image Server
- Scrittura

Set Up a Test Configuration

In certain environments, it can be required that you run or re-run parts of the process instead of the whole automated chain. This section details how the different modules operate separately.

Teleform Test Configuration

In the following test configuration, Teleform is not connected to a fax server. Users manually drop inbound documents in the Teleform drop box.

1. Start Teleform Reader, Teleform Scan Station, and Teleform Verifier.
2. Drop the inbound files (in PDF or TIFF format) into the Teleform drop box as configured in Configure the Polling Job, on page 336.
3. In the Teleform Designer, click **Utilities** and select **Batch Management Dialog**. The **Batch Management Dialog** box displays.

   Click Refresh to check the job progress.

   - If Batch Commit displays in the Status column, the job is complete.
   - If the job is stuck at a different state, right-click the job and select **Commit**.
4. Check the output folder defined in Designate Signatures for Matching for the output.

IDOL Image Server Test Configuration

In the following test configuration, the IDOL Image Server signature matching is initiated and receives the matching signature and confidence level.

1. Place the Teleform output in the folder specified by `@inputfiles` in `SignatureID.pl`.
2. Run `SignatureID.pl`.

   The output is placed in the folder specified by `$dirToOutput` in `SignatureID.pl`.

Scrittura Test Configuration

Scrittura consumes the output of the Teleform and IDOL Image Server processes and distributes the output into the workflow.

To manually simulate the output consumption and distribution to the workflow

1. Place copies of the IDOL Image Server output, the TIFF files containing the signatures, and the original inbound images in the drop box configured for Scrittura.

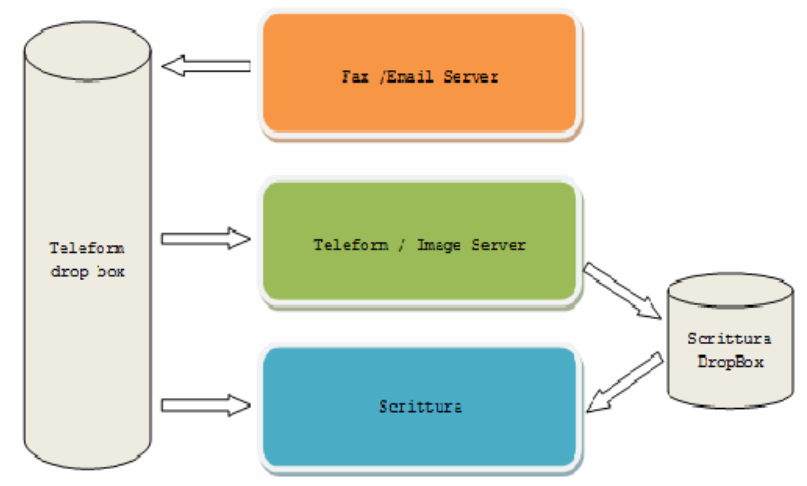
   These documents are stored in DocManager.

2. Place another copy of the output from IDOL Image Server in the message drop box configured for Scrittura.

   This is the inbound message that Scrittura processes.

General Deployment in a Production Environment

The following illustration summarizes the general deployment of the Scrittura OCR process.



The following topics provide steps to set up the environment once the different processes have been installed and configured.

- Start Teleform Components
- Start IDOL Image Server
- Start Scrittura

Start Teleform Components

Start the Teleform Reader, Teleform Scan Station, and Teleform Verifier. Incoming PDF and TIFF files come from fax or email servers.

Integration with external fax or email servers is outside the scope of this document. This document assumes that the relevant process is configured to place incoming files into the inbound Teleform drop box and the Scrittura inbound image drop box.

Teleform Reader can run as a Windows service when using the Enterprise edition.

Start Image Server

Once you have started the Teleform processes, you must start the IDOL Image Server.

To start the IDOL Image Server

1. From the desktop, open the Administrative Tools menu, select the **Services** facility.

   Start the DiSH license verification service.

2. Run `imageserver.exe` to start the IDOL Image Server service.

3. From Windows Administrative Tools, open the Task Scheduler.

   Select Create Task and configure the Perl interpreter to run `SignatureID.pl` once per minute.

The path variables in `SignatureID.pl` should match the rest of the configuration:

- `$teleformHotFolder` should match the Teleform input drop box.
- `@inputfiles` should be configured to look for files output by Teleform in the directory specified in Designate Signatures for Matching, on page 335.
- `$dirToOutputImages` should be set to the Scrittura inbound image folder.
- `$dirToOutput` should be set to Scrittura the inbound folder.

Start Scrittura

Start Scrittura using the typical process.

Upon receiving a new inbound confirmation, the Perl scrip `SignatureID.pl` sends the corresponding documents and XML metadata to Scrittura to be processed by the inbound workflow.

Signature of Inbound Documents

TIFF or image PDF inbound documents can be signed using Scrittura's Signature Applet.

The Signature Applet is a separate JAR file that must be deployed in the application.

A link to the Signature Applet should be added to the queue view to let the user load the applet. This is done using the `scrittura:signature` tag of the Scrittura tag library. For full details on this setup, see Signature Tag, on page 175.

Signatures used by the Signature Applet should be integrated into the application EAR, under `/custom-web/customJSPs/images/sign`.

Multiple signatures while a confirmation is being signed by a user are prevented. That behavior can be overridden by setting the startup option `scrittura.allowMultipleSigns` to true, as follows:

```
-Dscrittura.allowMultipleSigns=true
```

In order to improve performance of the signature applet and quality of the display, the TIFF document can be displayed on a gray scale by setting the startup option `scrittura.pdfpreview.quality` to grayscale, as follows:

```
-Dscrittura.pdfpreview.quality=grayscale
```

Chapter 14: Electronic Messaging

This section details the electronic messaging capabilities of Scrittura that allow integrating Scrittura with third-party platforms usch as, DTCC, ICE or SWIFT

This section contains the following topics:

Electronic Messaging Overview

The electronic messaging system in Scrittura provides the ability to send reliable, verifiable, and standardized messages between parties. The three systems currently supported by Scrittura are DTCC, ICE, and SWIFT.

These systems are loaded and run from within the workflow and can be configured to automatically send and handle incoming messages.

The following section explains the Scrittura side configuration for each messaging system. For more information regarding setting up the messaging systems, refer to relevant documentation (such as Trade Confirm, EnConnect, and so forth).

> **NOTE:** Electronic messaging capabilities are not bundled with the Scrittura platform. Check your license agreement for full information.

DTCC Messaging

The Scrittura Messaging Module is designed to provide a means of generating structured FpML messages conforming to a given messaging specification, such as DTCC Credit Default Swaps, and process updates sent by DTCC DerivSERV.

The data for each message is taken from the name/value map of the Product Instance, and then transformed as required to conform to the messaging requirements.

The DTCC Messaging Module converts a Scrittura trade into a DTCC message, which is subsequently sent to the DTCC DerivSERV platform. Updates sent by DTCC DeriSERV are then processed by the Messaging Module in order to update the trade in Scrittura. This section explains the process involved in adding the Messaging module to Scrittura and should be read in conjunction with the DTCC specification documents available on the DTCC website.

NOTE: DTCC can also be configured to run through EnConnect. For more information regarding EnConnect setup, see the EnConnect documentation.

Install DTCC Messaging

Given that messaging specifications frequently change, the Messaging module is designed to run as an optional component of Scrittura and therefore is not deployed as part of the main Scrittura installation. As a result, the messaging module can be updated without having to redeploy Scrittura.

To install DTCC Messaging

1. Expand scrittura3.ear file.

   Add the messaging.jar file to the /lib folder of the expanded `scrittura3.ear` enterprise `archive file`.

   Re-zip the expanded folder back into scrittura3.ear.

2. Create a TransformVariableConfig.xml file based on the details provided in Inbound Message Configuration for DTCC Messaging, on the next page.

   Copy the file to the Scrittura configuration folder.

   A sample version of `TransformVariableConfig.xml` file is provided with the distribution.

3. Perform the Outbound Configuration.
4. Update `TransformVariableConfig.xml` to reflect the mappings and appropriate transformations required to provide the selected message formats with the correct data. See `TransformVariableConfig.xml, below`.
5. Add the required steps to your workflow.
6. Perform the Inbound Configuration.
7. Add the required steps to your message processing workflow as described in Inbound Message Configuration for DTCC Messaging, on the next page.

Message Generation Configuration for DTCC Messaging

This section describes the message generation configuration files to configure DTCC messaging.

TransformVariableConfig.xml

The mapping between internal Scrittura variable names and the external Messaging variable names used must be defined in the configuration.

This is done using the `TransformVariableConfig.xml` file, located under the Scrittura configuration folder. For full details, see Variable Mapping Configuration, on page 350.

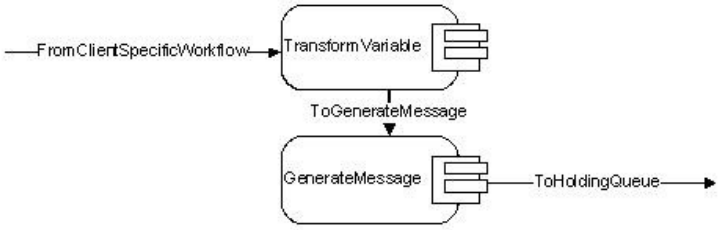
Custom Variable Transformation Classes

The DateTransformation class is the only transformation class that is provided as standard. If any other transformations are required then custom transform classes can be created by implementing

the `com.ipicorp.scrittura.messaging.TransformVariableFormat` interface.

Message Generation Workflow

For outbound messages, two classtools need to be added to your workflow at the point where outbound messages are to be generated. One further classtool may need to be added to customize the data used in variable transformation, such as to swap buyer and seller fields depending on some flag, and so forth.



Before an ActivityItem enters the TransformVariable class, the data required for the message type that is to be generated must be present in the ProductInstance. For details about those variables, see the documentation for the particular message type.

Once the message is generated by GenerateMessage the message is stored in DocManager in the folder for that particular ProductInstance. The filename assigned to the message is dependent on the message type used. If a message of that type and action already exists for the ProductInstance then a new version of that message will be stored.

Inbound Message Configuration for DTCC Messaging

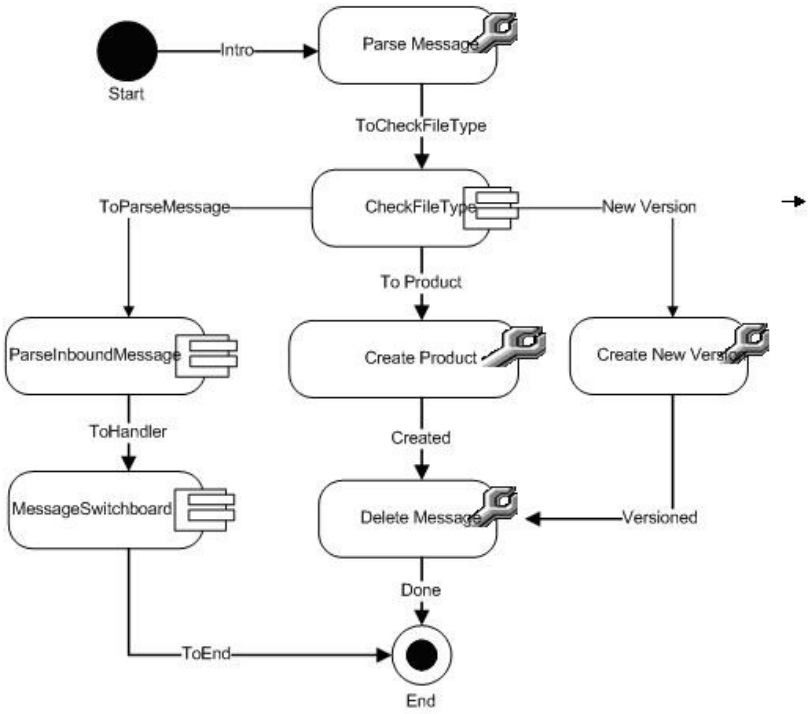
Responses received from the messaging service will arrive as standard through the `scrittura-tickets` JMS queue, whether having been picked up from a drop box or by some other means.

Inbound Message Workflow

The following illustration is a simple example of DTCC message processing workflow and does not represent a real workflow implementation. In this example; three additional class tools are added to the standard message processing workflow to detect messages, perform custom message parsing, and handle messages depending of their contents. All these class tools are present in the `com.scrittura.messaging` package.

The first class tool, `CheckFileType`, determines whether or not a message ticket contains a response from a recognized messaging provider.

`ParseInboundMessage` parses the message and extracts the data necessary to route the message using `MessageSwitchboard`.



`MessageSwitchboard` uses the factory specified as a parameter to WebLogic (the `MessageHandlerFactory`) to retrieve the appropriate `MessageHandler`.

Inbound Message Handlers

Once an inbound message has been received from the message service provider and identified as such, the `MessageSwitchboard` class tool provides the link into client-specific code for customized handling of such messages. To do this a factory class must be implemented along with several message handlers.

ICE Messaging

The ICE messaging system is implemented by using EnConnect and Trade Confirm (see the Trade Confirm documentation for more information). ICE has no configuration on the Scrittura side so the only configuration needed is for the EnConnect server.

The ICE messaging system will send a trade request to the ICE platform with the appropriate information from the Scrittura trade and will receive back a confirmation response from ICE. To retrieve messages from ICE it is necessary to set up a repeating action in EnConnect to poll their servers.

EnConnect must be added to the workflow using the `GenerateMessage` classtool (for more information, review the workflow documentation). This classtool will transform the PI being processed into canonical XML and send it to the specified EnConnect server. EnConnect will send back a response in a similar canonical XML format populated with the relevant information from ICE.

The server is configured using the following extended attributes.

Attribute	Type	Description
onProcessingExceptionMoveToErrorState	Boolean	Move to the error state when there is a problem with the message generation.
storeErrorsInVariable	Boolean	Store the error state/message in the error variable.
errorsVariableName	String	Name of the variable to store error messages in.
messageFormat	String	The type of message (such as dtcc or ice).
connectionFactoryName	String	The name of the connection factory.
jndiInitialContextFactoryName	String	The name of the initial jndi context factory.
serverUrl	String	The URL url for the EnConnect server.
topicName	String	The name of the topic.
queueName	String	The name of the EnConnect queue.
Transacted	Boolean	Whether EnConnect is using transactions or not.

SWIFT Messaging

Scrittura contains built-in classes for both SWIFT message generation and SWIFT message parsing.

The message generation is performed using a Scrittura class tool which can be added as an item in a Scrittura workflow. This classtool calls TradeConfirm, a separate server, which performs the actual message generation. TradeConfirm supports a subset of the SWIFT message range.

The SWIFT message parser implements the Scrittura interface for parsers which are configured in `scrittura-config.xml`. If configured, every trade dropped into Scrittura will be tested to see whether it conforms to the SWIFT message format, and if so it will be parsed and a map of variables generated from its contents.

Both message parsing and generation make use of an external server, Trade Confirm, which performs the actual message processing (generation and parsing). The setup of this server is detailed in its own technical note. The remainder of this section details the configuration to perform on Scrittura side in order to integrate SWIFT messaging to Scrittura.

General Configuration for SWIFT Messaging

SWIFT messaging is configured using the following files.

- `spring-config-http.xml`
- `TransformVariableConfig.xml`

- scrittura-config.xml

spring-config-http.xml

The spring-config-http.xml configuration file controls the communication with Trade Confirm using spring remoting. This file is located under the Scrittura configuration folder.

This Spring file should contain a bean, tradeConfirmService, with the lazy-init attribute set to true. It should also have the serviceUrl and serviceInterface properties defined.

Example

```
<bean id="tradeConfirmService" class="org.springframework.remoting.httpinvoker

.HttpInvokerProxyFactoryBean"

lazy-init="true">

<property name="serviceUrl"

value="http://[host]:[port]/generation.service"/>

<property name="serviceInterface"

value="com.autonomy.tradeconfirm.services

.TradeConfirmationService"/>

</bean>
```

Set the host and port in the serviceUrl property of the tradeConfirmService bean to point to the Trade Confirm server.

> **NOTE:** This Spring file uses XSD validation rather than DTD validation.

TransformVariableConfig.xml

The mapping between internal Scrittura variable names and the external Messaging variable names used must be defined in the configuration.

This is done using the TransformVariableConfig.xml file, located under the Scrittura configuration folder. For full details, see Variable Mapping Configuration, on the next page.

Inbound Message Configuration for SWIFT Messaging

A new parser has to be added to scrittura-config.xml in order to parse incoming SWIFT messages into usable variables in Scrittura. Add the following entry to the list of parsers in scrittura-config.xml:

```
<message-type name="SWIFT"

class="com.ipicorp.scrittura.tradeconfirm.parsers

.SWIFTParser" />
```

This parser will send all incoming messages that match the SWIFT message schema to the Trade Confirm server. Trade Confirm will be responsible for the actual transformation of the SWIFT

message into a map of Scrittura variables. The result will be sent back to the parser which will use it to populate the Message Ticket of the item in the Message Processing workflow.

Message Generation Configuration for SWIFT Messaging

Add the following class tool wherever appropriate in the workflow configurations:

`com.ipicorp.scrittura.tradeconfirm.classtools.SWIFTMessageGenerator.`

This class tool requires a single extended attribute to be set, documentTitle. This attribute determines the name of the file to be attached to the appropriate Product Instance following successful SWIFT generation.

Note that the class tool can only be attached to a workflow where the Product Instance has already been created.

Variable Mapping Configuration

Variable mapping is necessary for DTCC and SWIFT messaging in order to map Scrittura PI variables to the external variables used by either `messaging.jar` (for DTCC) or Trade Confirm (for SWIFT). A similar mapping is required for ICE but takes place within EnConnect, hence is outside the scope of this document.

Variable mapping is done using the `TransformVariableConfig.xml` file located under the Scrittura configuration folder. The message-transform module must be enabled in `startup-config.xml`.

The `TransformVariableConfig.xml` file consists of a series of TradeType tags each of which has a type attribute. The type attribute must correspond to a valid Trade Confirmation Type (DTCC, ICE, or SWIFT).

Each `TradeType` tag consists of a large number of Variable tags. Each Variable tag represents a mapping between a Trade Confirm variable and a Scrittura variable. Each Variable tag may contain the following attributes.

Attribute	Required/ Optional	Description
internalVariableName	Required	Name of Scrittura variable.
externalVariableName	Required	Name of Trade Confirm variable.
dataType	Required	Variable type; only exists for use by `transformClass`
oldFormat	Required	Can be left blank; only exists for use by `transformClass`
newFormat	Required	Can be left blank; only exists for use by `transformClass`
transformClass	Required	Allows variable value manipulation; can be left blank.
transformationType	Optional	Determines whether this mapping should be performed for message generation only, message parsing only, or all usages of the configurations. `generation, parsing, or all (default)`

Each transformation class should be referenced by its full package name and must be an implementation of the `TransformVariableFormat` interface accessible within the Scrittura classpath. This interface is located in the `trade-confirmation-client.jar` located in the `/lib` folder. It contains a single method, `transformVariable`. This method has two inputs, an object holding all the information read from the Variable tag's attributes and the input value to be transformed, and returns the transformed value as a String.

Note that this mechanism can be used to add additional values by referencing input variables. However, in such cases the `transformationType` of the variable should be specified so that it only applies to what it is intended to (generation or parsing).

The following example is a straight forward mapping with no transformation.

```
<Variable internalVariableName="dtcc_product_type" externalVariableName="dtcc_
productType" dataType=""

oldFormat="" newFormat="" transformClass=""/>
```

The following example defines the mapping to the `dtcc_terminationTradeData` variable from the `maturity_date` variable. Also note that the expected format of `maturity_date` is `dd mmm yyyy` and that the variable is stored as a String.

All dates used in DTCC messages must be specified in the format "yyyy-MMM- dd" so the DateTransformation class is used to convert between the two formats.

```
<Variable internalVariableName="maturity_date" externalVariableName="dtcc_
terminationTradeDate" dataType="String"

oldFormat="dd mmm yyyy" newFormat="yyyy-MM-dd" transformClass="DateTransformation"/>
```

For a list of variables required for the message type to be utilized see the DTCC documents for that specific message type available from the DTCC website.

Chapter 15: Structured Products

This section details the handling of Structured Products in Scrittura and how to implement such products.

This section contains the following topics:

Structured Products Overview

A Structured Product (or Structure) is a derivative instrument made up of a collection of trades (Components) linked together for economic reasons.

Example

A Dual Currency Deposit (DCD) is a short-term fixed coupon investment redeemed in a foreign currency. As such it consists of a Deposit and one or more FX Options, which cover the foreign exchange rate risk.

Structures can have any number of components. The following are typical confirmation scenarios.

- Components are confirmed individually (1 confirmation issued per component).
- Components are aggregated into different groups within the Structure, each group being confirmed separately (1 confirmation issued for each group).
- The Structure is confirmed as a whole (1 confirmation issued for the whole Structure).

Scrittura allows full management of Structures and to the ability to STP Structures or groups of components with no manual intervention.

Confirmation Groups

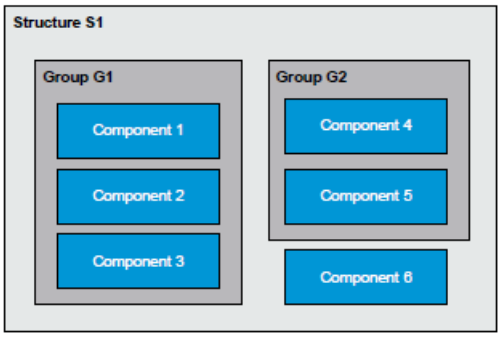
Groups of components can be defined within a Structure for confirmation purposes. Groups (also called Confirmation Groups) are an essential concept for the confirmation of Structures. They define how confirmations will be generated and handled within the Structure. Groups can only be defined within the same Structure and cannot span different Structures.

Confirmation groups can be created automatically, by applying rules, or manually. Components can be removed from a group (ungroup action) as required.

Example:

Structure S1 has 6 components (Components 1 to 6) and two groups (G1 and G2) have been defined within the Structure.

- Group G1 encompasses Components 1, 2, and 3.
- Group G2 encompasses Components 4 and 5.
- Component 6 does not belong to any group within the structure.



In this example, three confirmations are generated.

- One confirmation for Group G1.
- One confirmation for Group G2.
- One individual confirmation for Component 6.

Structure Handling in Scrittura

There are a number of different scenarios in which Structures may be employed in Scrittura.

- Fully qualified Structures may be created by the upstream systems, in which case components are pre-populated with a known Structure Reference.
- Trades are marked as components belonging to a Structure, although the Structure itself is not defined (no Structure Reference available). These components are called Orphan Components.
- The Structure is not created upstream and trades are not marked as components but a set of standalone trades needs to be defined as a Structure in Scrittura.

When trades are not linked within a defined structure, it is possible to link them together in Scrittura as a Structure (link operation). Conversely, trades belonging to a Structure can be removed from their original Structure and injected into the workflow as standalone trades (unlink operation).

Components typically reach Scrittura as a series of separate XML messages— one per component—or within one large XML message that encompasses all components. Once received by Scrittura, a Scrittura Product Instance (PI) is created for each component that gets stored in a **Component Store** queue.

Subsequently, during the confirmation group creation process, a stub is created for each group that will drive the confirmation process throughout the workflow, while the actual components within the group remain in the Component Store.

Structured Product Configuration and Setup

This section describes how to configure structured products in Scrittura.

Structured Product Configuration

The main configuration file for Structured Products is `structured-product- config.xml` and is located in the Scrittura configuration folder.

The configuration file contains several sections, represented by the following nodes:

- key-values Node, below
- stub-creation-variables Node, on the next page
- status-event-mapping Node, on page 356

key-values Node

The `key-values` node of the Structured Products configuration file contains the key configuration fields, defined by the following child nodes.

Child Node	Required/ Optional	Description
struct-proddef-shortname	Optional	Product definition short name for structured products. Default: `STRUCT`
event-processing-mode	Optional	Set Component to process events at component level or to Structure to process events at Structure level. Default: `Component`
outbound-workflow	Optional	Workflow containing the Component Store and Holding Queue. Default: `standardOutbound`
component-store	Optional	Name of the queue used as the Component Store. Default: `ComponentStore`
holding-queue	Optional	Name of the Holding Queue used for event processing. Default: `HoldingQueue`
struct-ref-variable	Required	PI variable containing the Structure Reference ID.
parser-component-xpath	Optional	XPath of the root component nodes used to identify separate components when the input message sent to Scrittura takes the form of a single XML file.

Example:

```
<key-values>
<struct-proddef-shortname>STRUCT</struct-proddef-shortname>
<event-processing-mode>Component</event-processing-mode>
<outbound-workflow>standardOutbound</outbound-workflow>
<component-store>Component Store</component-store>
<holding-queue>HoldingQueue</holding-queue>
<struct-ref-variable>tradeLinkID</struct-ref-variable>
<parser-component-xpath>/FILE/ITEM</parser-component-xpath>
</key-values>
```

stub-creation-variables Node

The `stub-creation-variables` node of the Structured Product configuration file defines the mapping between stub variables and component variables in order to populate the stub upon creation. The `stub-creation- variables` node takes one attribute, custom-data-handler, which is optional and, where provided, must contain the explicit name of a Java class to be used for extending the default mapping procedure.

The child nodes of `stub-creation-variables` are `<mapping>` elements which must have an empty body. `<mapping>` has the following attributes.

Attribute	Required/ Optional	Description
stub-variable	Required	Name of the variable in the stub.
component-variable	Required	Name of the component variable where the value to populate the stub is taken.
component-id	Required	Identifier of the component it applies to.

Example:

```
<stub-creation-variables
custom-data-handler="com.ipicorp.scrittura.CustomDataSample">
<mapping stub-variable="tradeLinkID" component-variable="tradeLinkID" component-id="1" />
<mapping stub-variable="counterpartyName" component-variable="counterpartyName" component-id="1" />
<mapping stub-variable="tradeDate" component-variable="tradeDate" component-id="1" />
</stub-creation-variables>
```

status-event-mapping Node

The `<status-event-mapping>` node defines the status and event mapping table.

The `<status-event-mapping>` node has the following attributes.

Attribute	Required/ Optional	Description
event-type-variable	Required	Defines the name of the variable holding the trade event type.
trade-status-variable	Required	Defines the name of the variable holding the trade status.

The `<status-event-mapping>` node must contain `<mapping>` elements that have an empty body. `<mapping>` has the following attributes.

Attribute	Required/ Optional	Description
mapping-type	Required	The event type. Possible values are `Held, New, Cancel, Amend`.
event-type	Required	Value of the event type variable for this event.
status	Required	Value of the status variable to be set for this event.

Example:

```
<status-event-mapping
event-type-variable="eventType" trade-status-variable="tradeStatus">
<mapping mapping-type="Held"
eventType="" status="Held" />
<mapping mapping-type="New"
eventType="" status="Live" />
</status-event-mapping>
```

Product Definitions

Structured Products use a set of pre-defined or configurable variables, which must be declared in the corresponding Product Definitions for both stubs and components.

Some variables are specific to components, others to stubs, and some common to both stubs and components. Hence three Product Definitions are provided with the distribution.

- `struct-component-vars.xml`. Sub-Product Definition that contains variables required for components, which must be included into all Product Definitions of products that can potentially

be structure components (such as, EQD, FXO).

- `proddef-structure.xml`. Product Definition that contains variables required for structure stubs.

  > **NOTE:** The short name in proddef-structure.xml should correspond to the one defined in structured-product- config.xml.

- `struct-common-vars.xml`. Sub-Product Definition that contains variables required for both components and stubs. This sub-Product Definition must be included into all Product Definitions of products that can potentially be structure components and also in the stub Product Definition.

Those Product Definitions should be placed under the `\products` directory in the Scrittura live folder. As for any Product Definition, the Structure Product Definitions will be added to `scrittura-config.xml` and will contain all necessary variables required by the business. The stub variables defined in structured-product-config.xml should be declared in the structure stub Product Definition, `proddef-structure.xml`.

The Product Definition for structure stubs is proddef-structure.xml, whereas components retain their original Product Definition (such as, FXO, EQD) as if they were standalone trades, except that those must include `struct-common-vars.xml` and `struct-component-vars.xml`.

Stub Variables

The following variables are mandatory for the structure stub and are used internally by Scrittura.

Variable Name	Type	Description
scrGroupAmendTimestamp	String	Timestamp of an event triggered on a confirmation group.
scrComponentAmendTimestamp	String [ ]	Array of individual timestamps for each component.
scrComponentEventType	String [ ]	Array containing the current event type of components.
`scrNumberOfComponents`	Integer	Number of components belonging to the group.
`scrMasterComponentID`	String	`CommonReferenceID` of the Master Component.
`scrComponentID`	String [ ]	Array containing component `CommonReferenceID` values, orders as per the component ordering.

Component Variables

The following variables are mandatory for each component and are used internally by Scrittura.

Variable Name	Type	Description

scrComponentOrder	Integer	Integer that holds the order of a component within the confirmation group.
scrIsMasterComponent	Boolean	Set to true if this is the Master Component; set to false if it this is not the Master Component.
scrAmendComponentNow	Boolean	Internal flag to trigger event processing on a component. Do not alter.

Structured Products Common Variables

The following common variables are mandatory for each component and stub and are used internally by Scrittura.

Variable Name	Type	Description
scrIsStub	Boolean	Set to true if the PI is a structure stub; set to false otherwise.
scrIsComponent	Boolean	Set to true for trades that are components of a structure.
scrIsGrouped	Boolean	Set to true if the PI belongs to a confirmation group.
scrGroupID	String	Unique reference that identifies a confirmation group.

Message Parsing

Structures can reach Scrittura in many ways, depending on how they are specified in the upstream system.

The following are examples of ways Structures reach Scrittura.

- Fully qualified structures (known type, defined list of components, and known Structure Reference) are sent to Scrittura as a single message.
- Fully qualified structures are sent to Scrittura as multiple independent messages.
- Orphan components belonging to an unknown Structure are sent to Scrittura as multiple independent messages, and linked together in Scrittura.

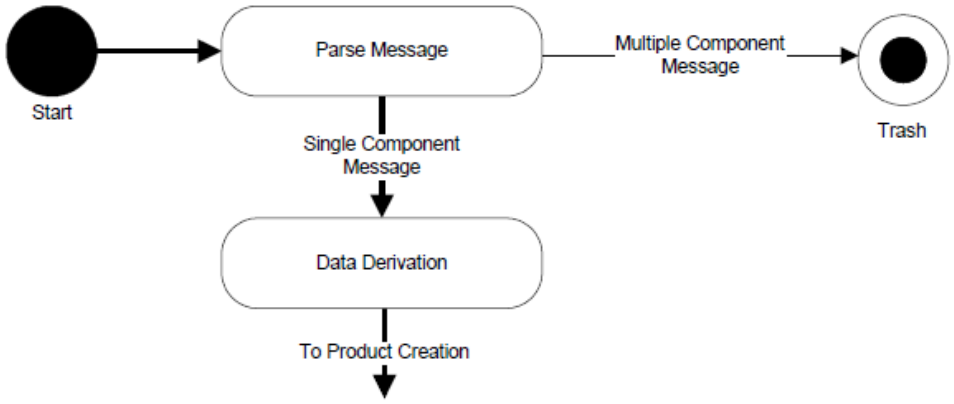
A generic XML Parser for Structures is provided in Scrittura. Define the required CSV file for the parser and add the generic parser class to scrittura-config.xml using the following as an example:

```
<message-type name="StructXML"

class="com.ipicorp.scrittura.messages

.GenericXMLStructureParser" />
```

Aggregated messages where all components are sent in a single XML message are cut down into individual messages and subsequently re-injected into the workflow. The initial aggregated message must then be discarded, as depicted in the following illustration.



For details on how to set up the parser for incoming Structured Product messages, see Structured Product Message Parsing, on page 287.

Workflow Setup and Event Handling

This section provides an illustration as to how structured products can be integrated into the Scrittura Message Processing and main outbound workflow, It provides an overview as to how business events can be handled for structured products.

Event Processing Mode

Just as for standalone trades, Structures can receive events and market operations from upstream systems which act upon Structures. Events can be handled in Scrittura at component or structure level, configured in the event- processing-mode field in structured-product-config.xml.

Component Level

If the event processing mode is set at component level, an event is triggered in Scrittura upon reception of an event for any component.

The component is amended in the Component Store where it sits. If this event belongs to a confirmation group, the stub is amended as well and the event is processed from the start of the workflow like any other event.

Event type and status for the stub are determined by the status and event mapping configuration defined in structured-product-config.xml.

- If the event types of all components are New, the stub event type will be `New`.
- If the event types of all components are Cancel, the stub event type will be `Cancel`.
- In all other scenarios, the stub event type will be `Amend`.

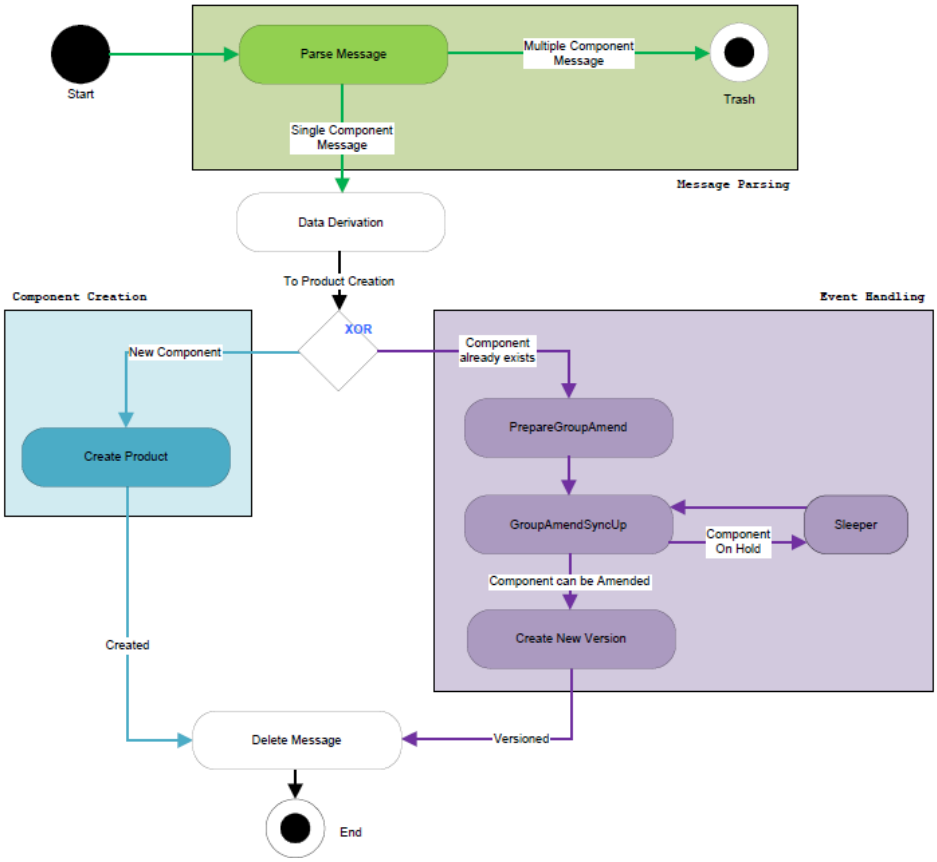
Structure Level

When the event processing mode is set at Structure level rather than component, the event is triggered in Scrittura once all components have been amended.

The remainder of the process is identical to the Component Level mode.

Message Processing Workflow and Event Handling

The following illustration depicts an example of a Message Processing workflow that handles Structured Products.



Message Parsing

Message Parsing is the first process in the workflow and is discussed in detail in Message Parsing, on page 358.

Data Derivation

Data Derivation is a common message processing workflow step that consists of deriving variables from the existing set of variables, by way of a series of rules. BLogic can be used in this capacity.

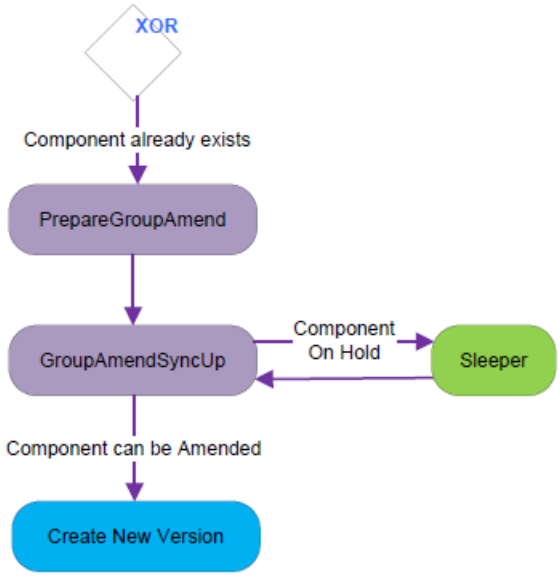
Component Creation

Component Creation creates components in Scrittura whenever the component does not already exist.

Event Handling

The Event Handling portion of the workflow is responsible for handling business events related to a trade.

The following illustration is a subset of the Message Processing Workflow, highlighting event handling.



Prepare Group Amend

If the incoming Message Ticket relates to a component that is part of a group, the Prepare Group Amend classtool alerts the stub that an event has been triggered and prepares the stub for processing. Prepare Group Amend moves the stub from its position in the workflow to the Holding Queue, if not already there, changes its status to Held, and sets required internal variables.

Classtool	`PrepareGroupAmend`
Package	`com.ipicorp.scrittura.structures.messageprocessing`
Attributes	None

This classtool performs no action if the Message Ticket relates to a standalone trade or a component that is not part of a confirmation group.

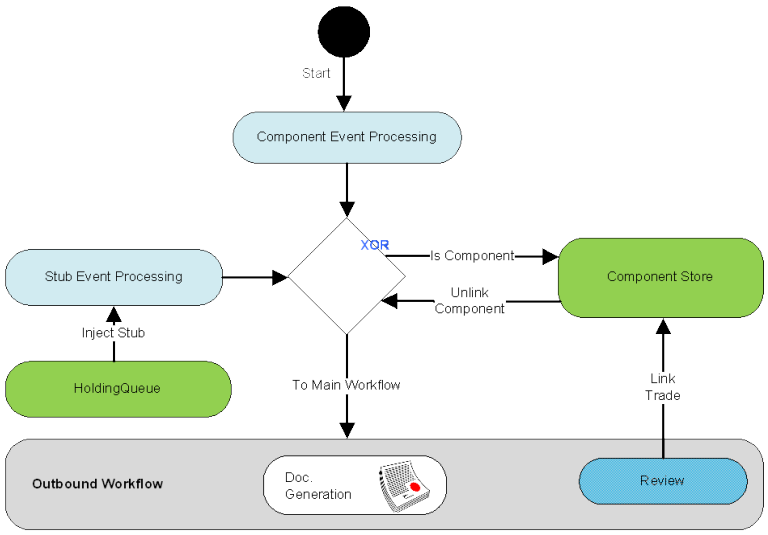
Group Amend Sync-Up

When the incoming Message Ticket relates to a component that is part of a group, the Group Amend Sync-Up classtool ensures that the Group PI has successfully reached the Holding Queue before releasing the Message Ticket on to the Version Creation step. In the meantime, the Message Ticket is held in the Sleeper activity, a simple Scrittura timer.

Classtool	`GroupAmendSyncUp`
Package	`com.ipicorp.scrittura.structures.messageprocessing`
Attributes	None

Outbound Workflow and Event Handling

The following illustration depicts an example of the part of an outbound workflow specific to Structured Products.



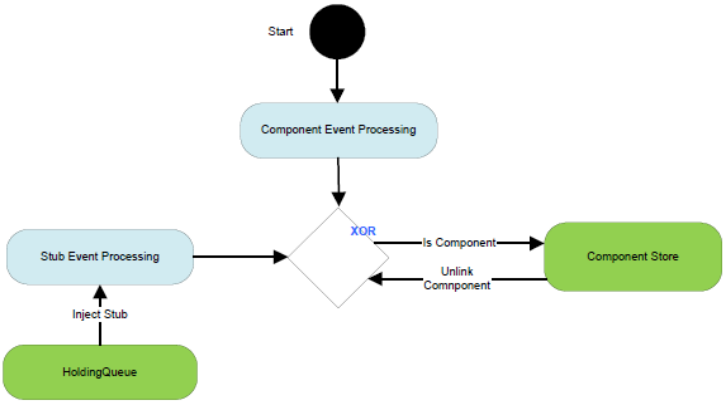
Component Store

The Component Store is a manual queue where components are stored after entering the main workflow. A transition must lead to this queue whose condition is based on the variable `scrIsComponent` with a value of `true` (`scrIsComponent=true`).

Link Trade and Unlink Component Transitions

Link Trade and Unlink Component transitions exist if linking and unlinking operations are allowed. When provided, these operations cause trades to be moved in and out of the Component Store.

The following illustration is a subset of the outbound Workflow, highlighting event handling.



Holding Queue

Stubs are held in the manual Holding Queue while an event relating to them is being processed. Event processing for Structures may encompass multiple individual events at component-level.

Component Event Processing

If event processing is at component level, the processing of the event by the Component Event Processing classtool triggers the release of the stub from the Holding Queue. If it is at structure level, release of the stub occurs when the last component event has been processed.

Classtool	`GroupAmendHandler`
Package	`com.ipicorp.scrittura.structures.outbound`
Attributes	None

Stub Event Processing

Once released from the Holding Queue, the stub is processed by the Stub Event Processing classtool, which finalizes and terminates the event processing. The amended stub is at the beginning of the workflow, available for processing whenever the next event is received.

Classtool	`GroupAmendProcessing`
Package	`com.ipicorp.scrittura.structures.outbound`
Attributes	None

Linking and Grouping

Linking and grouping, along with their counterpart operations, unlinking and ungrouping, are essential capabilities for the handling of Structures in Scrittura.

Trade Linking

Trade linking consists of linking trades together under the same structure. Trades are linked to a Structure by assigning them a Structure Reference. The Structure Reference is held by a PI variable, which can be configured in `structured-product-config.xml`. For a component, the Boolean variable scrIsComponent must also be set to true.

A trade is considered as part of a Structure if its Structure Reference is not null or empty, and standalone if its Structure Reference is empty or null. The variable `scrIsComponent` must always be kept in sync with Structure Reference being populated or be empty.

The reverse operation, Trade Unlinking, involves removing the Structure Reference value from the trade and setting the value of `scrIsComponent` to `false`.

Component Grouping

Component Grouping consists of grouping together a number of components belonging to the same structure for confirmation purposes. A single confirmation is issued for the whole group and that confirmation may quote data from the different components of the group.

When grouping components, a Master Component is designated and components are ordered within the group. A stub is then created for the group and injected into the workflow. The stub is a standard PI that links the components together and drives the confirmation process in the workflow.

Components can be removed from a group, and additional components can be added to a group.

Component Ordering

An order must be assigned to a Component, to be referred to in the Product Views for document generation purposes.

Master Component

The Master Component (or Main Component) of the group is specified during this operation as being the first component of the ordered group. The Master Component can be used for different purposes, such as a default display for the group or to define the Group ID. All sibling components have the variable `MasterComponentID` populated with the `CommonReferenceID` of the Master Component. The Master Component itself has the Boolean variable `IsMasterComponent` set to `true`.

Stub Creation

When created, other internal and user variables are populated in the stub. Internal variables are the variables necessary for Scrittura to handle Structures and are included in `struct-vars.xml`. For more information about stub variables, see Stub Variables, on page 357.

User variables are configurable variables (such as those common to all components, like counterparty information) and can be configured in `structured-product-config.xml`.

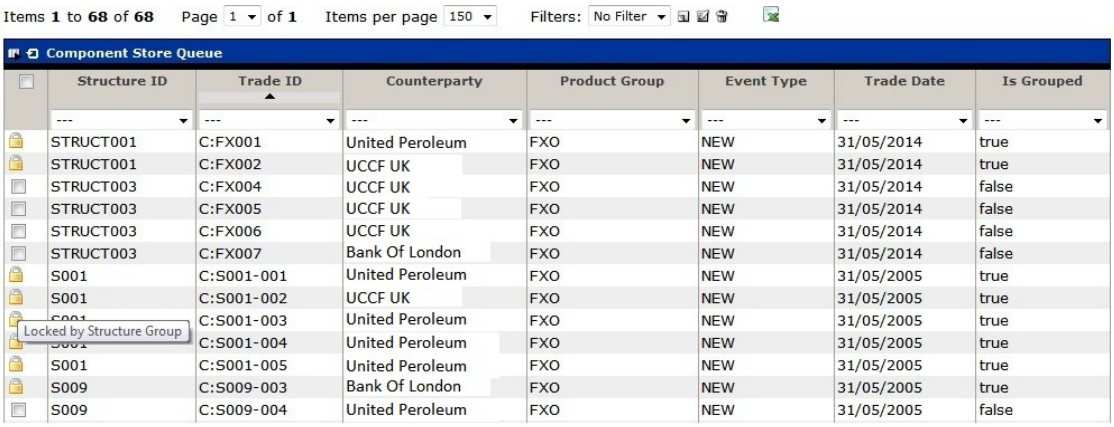
User Interface

Trade linking and component grouping can be handled by the upstream system and automated in Scrittura as part of a custom implementation. Scrittura provides the corresponding manual events and JSP screens in its core user interface.

Structure and Group management operations are provided as panels that can be integrated directly into Scrittura bulk screens. Configuration takes place in `general-ui-config.xml`, as `<panel>` nodes under the `<bulk-panels>` section. For each relevant case, page and `bulkTradeHandler` attributes of the panel node are provided.

Component Store

The Component Store screen is based on the bulk screen. All available components are displayed in this screen and results can be filtered out using column filters or general filters. A padlock (and tooltip) displays instead of a check box for trades already part of a confirmations group, as depicted in the following illustration.



The base JSP for this Component Store page is `/jsp/bulkBase.jsp`. The required panel names are passed as parameters. For more information, see Bulk Screen Configuration, on page 187.

Trade Linking

Trade linking is the only operation that is not available from the Component Store queue, but is accessible from standard queues. It is available from standard queues because trades involved in this process are standalone trades at this stage and are not components.

page	`linkComponentPanel.js;`
processHandler	`com.iwov.gcm.scrittura.web.queue.bulkactions.StructLink`

Trade Unlinking

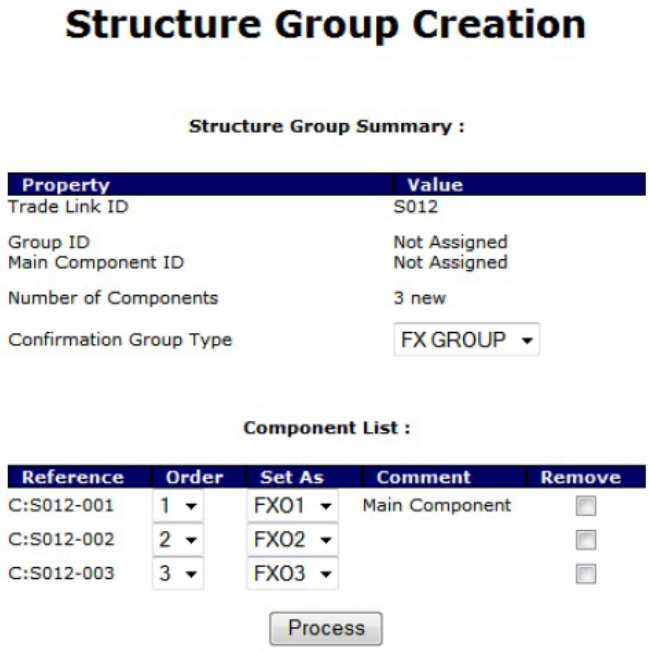
Unlinking trades consists in removing a trade from a structure so that it becomes a standalone trade, and gets re-injected into the workflow as such. This operation is available from the Component Store screen.

page	`unlinkComponentPanel.jsp`
processHandler	`com.iwov.gcm.scrittura.web.queue.bulkactions.StructUnlink`

Component Grouping

The user must specify a set of components and click the Group button in the Component Store Bulk Action panel. Optionally, a Group ID can be specified.

If the group does not exist or no group ID is provided, a new group is created. Otherwise the selected components are added to the existing group.

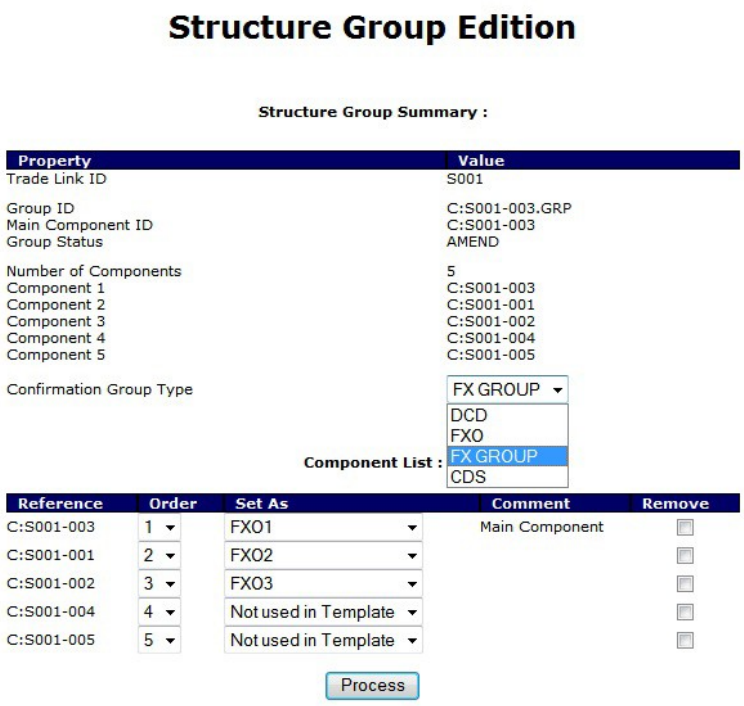


Component Editing

As with the group creation operation, the group edition panel leads to the group management console where component ordering can be re-arranged and basic group properties (such as confirmation group type) set. Components can also be removed from the group.

page	`editStructurePanel.jsp`
processHandler	`com.iwov.gcm.scrittura.web.queue.bulkactions.StructGroupEdit`

The following illustration depicts the editing of a Structure group.



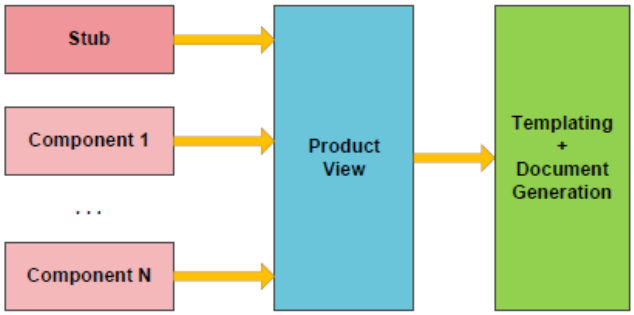
Document Generation for Structures

With the Document Generation Suite, it is possible to design templates for Structures as easily as for standalone trades. The ability to STP Structures is a direct consequence.

This section assumes that you are already familiar with the *Document Generation Suite*. Refer to the *Document Generation Suite* documentation for complete information.

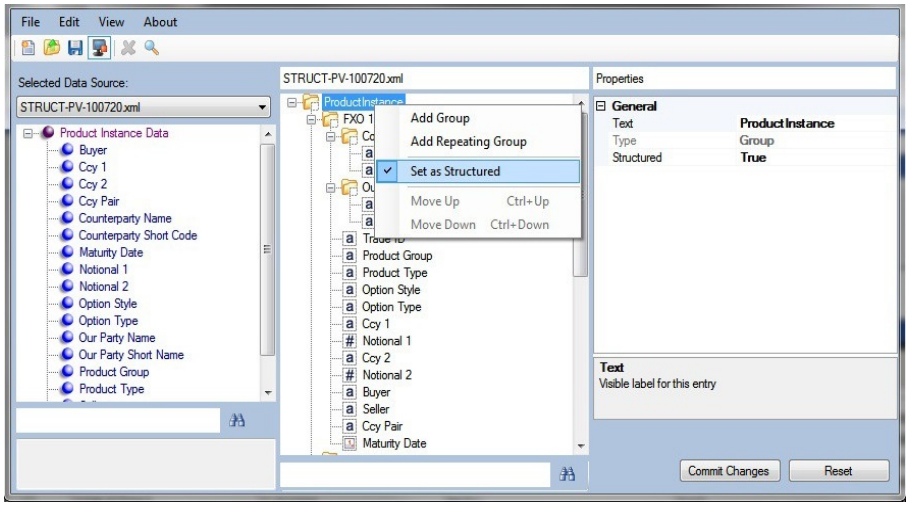
Product View Design

Structures are confirmed per group. Once Confirmation Groups are identified, the Product Views corresponding to those groups can be designed using the Product View Builder. A Product View is the actual data dictionary, fully designed and maintained by template authors. It contains all the economic data involved in template design along with the corresponding display and format information. In the case of Structures, the Product View aggregates data from the different legs, providing a single economic view for template design.



Product Views used for structure groups must be flagged as such using the Product View Builder; select the option “Set As Structured” on the Product View root node for this purpose.

In this mode, the top level of the Product View can contain only groups—no variables or repeating groups. Each of the direct child nodes of the root represents a component of the structure or the stub itself.



Nodes representing components in the Product View receive an identifier from 1 to the number of components, and the stub receives the identifier 0. The stub is optional in the Product View.

The order of components defined in the Product View corresponds to the order of the components within their group.

Template Design and Document Generation

Template design and document generation for Structures are exactly the same as for standalone trades.

Once the Product View is complete, it can be used to design templates in exactly the same manner as for standalone trades, since the product View hides the complexity of component aggregation.

Similarly, the Document Generation process uses the usual set of classtools and does not require further configuration.

Chapter 16: Scrittura Administration and Run-Time

This section describes the administration features offered by Scrittura, within its user interface or using its external administration tools. It also explains essential run-time operation in order to administrate, tune and monitor the platform.

This section contains the following topics:

Scrittura Administration Console

The Scrittura administration console lets you perform all Scrittura administrative tasks in an intuitive manner while providing monitoring capabilities.

The Scrittura administration screens include the following tabs.

- General
- Database Operations
- Advanced
- Monitoring
- Check Configuration
- Service Pack

Each tab is divided into expandable sub-panels that segregate the administrative features into functional groups.

Access the Scrittura Administration Console

The Scrittura administration console is accessible from the second-level Administration menu of the application, or from the following URL:

```
http://{servername}:{port}/scrittura/controller?e=configLoad
```

You must belong to the admins group and be logged in to the application in order to access the administration console.

General Tab

The General tab includes all of the common tasks which may need to be run while administrating Scrittura. These include starting and stopping both Scrittura and the workflow, reloading the three main module's configurations and reloading all of the configurations at once, individually or in a custom group.

Features are grouped into the following expandable panels.

- Start/Stop Scrittura and Workflow
- Reload Files

Start/Stop Scrittura and the Workflow Panel

Using the action buttons in the **Start/Stop Scrittura and the Workflow** panel on the **General** tab of the administration console, you can start or pause Scrittura and Workflow or reload the configuration.

- Click the **Pause/Resume Scrittura** button to pause or resume the Scrittura system. When paused, items within Scrittura are limited and many may be inaccessible.
- Click the **Pause/Resume** Workflow button to pause or resume the Workflow. When paused, the workflow does not process any trades. This means that trades do not move through the workflow and that any new trades are not processed until the Workflow is resumed.
- Click the **Reload Configuration** button to completely reload of all of the Scrittura, DocManager, and workflow configuration files (this process is also referred to as the SetConfig process). This can be customized to a degree by altering the startup configuration in startup-config.xml as detailed in SetConfig Process, on page 373.
- Click the **Reload Scrittura** button to only reload all of the Scrittura configuration files and folders into DocManager and into memory so that they are available to the application. This includes the startup- config.xml file which contains the configuration of optional modules.
- Click the **Reload DocManager** button to only reload all of the DocManager configuration files and folders into DocManager and into memory so that they are available to the application.
- Click the **Reload Workflow** button to only reload all of the Workflow files.

Reload Files Panel

The Reload Files panel on the General tab of the administration console lets you control the reloading of individual configuration files and folders into DocManager and into memory. When these files are reloaded the associated configuration class is reloaded and activated.

Select any combination of files and folders listed and click **Reload**.

> **TIP:** If you want to reload all files and all folders, use the Reload Configuration action button in the Start/Stop Scrittura and the Workflow panel on the General tab.

Database Operations Tab

The **Database Operations** tab includes the controls for database operations. You can also configure some of the Scrittura settings that can be altered at runtime.

This tab provides the DDLs (Data Definition Language) for generating the Fast Access tables (FA Tables), Archive tables, and Static Data Framework tables.

If the DDL is not displayed, click Show SQL. Right-click in the DDL content pane and select Copy to capture the DDL. You can then paste the copied DDL into your SQL client and run it.

Advanced Tab

The **Advanced** tab contains advanced Scrittura settings and equations.

The Scrittura settings let you adjust the trade throttles and JSP auto-reload time without having to run the `SetConfig` process or altering the configuration files.

Features are grouped into the following expandable panels.

- **cronjobs.** Provides the details of the cronjobs currently configured on the system and includes the status and schedule of each. The cronjobs can be tasked to run immediately, enabled, or disabled. Cronjobs are only visible if at least one cronjob has been defined.
- **Trade Throttles.** Adjust the trade throttles for queues.
- **JSP Auto-Reload.** Set the time interval, in seconds, for when the JSP auto-reload runs.
- **Equations.** Create, preview, and save equations that have been entered in the LaTeX format.

> **NOTE:** Unless the relevant changes are also made to the configuration files, all settings performed from this screen are reverted when a SetConfig process runs.

Monitoring Tab

The **Monitoring tab** includes all of the performance and workflow monitoring tools. These tools are to help the administrator find bottlenecks in the workflow and help diagnose issues.

Features are grouped into the following expandable panels.

- **Sessions.** Provides details about the current active sessions.
- **Workflow Reconciliation.** Includes workflow error reconciliation features that can be used to requeue trades when issues are detected in the workflow. The following reconciliation features are included.
    - **Find Stalled Items.** Find items that have been stalled for at least the defined amount of time (in seconds). The value entered should be relevant to the current workflow activity. In some cases of high activity, trades processing may take longer with trades not being stalled but just waiting for other activities to finish. A value of at least 1 hour (3,600 seconds) is generally recommended.

You can then requeue or delete the stalled item, or change the item state. It is recommended to first try to requeue the item. If the item does not recover after requeuing, change the state of the item to ERROR and treat the item as a normal ERROR item. Delete stalled items only as a last resort.

- **Find Invisible Items.** Find trades that do not have a work item associated to them, and therefore do not appear in the workflow.
- **Find Multiple Work Items.** Find trades that have multiple work items associated to them.
- **Find Orphan Work Items.** Find work items that do not belong to any trade.
- **Find Orphan Activity Items.** Find activity items that do not belong to any trade.
- **Find Orphan Audit Records.** Find audit records that do not belong to any trade.
- **Search Trades.** Find trades that have reached Scrittura within the defined timeframe.
- **Lookup Trades.** Find a specific trade by defining its internal Scrittura identifier, CommonReferenceID.
- **Lookup Trade by Workitem ID.** This feature allows searching for trades by specifying their Workitem ID.

- **Performance Monitoring.** allows the administrator to run some tests to determine the performance of the system based on several metrics that are chosen when it is run.

Check Config Tab

The Check Config tab provides configuration validation. The included tables provide notification of configuration issues and should be consulted if any such issues arise.

There is a table for each of the main Scrittura components, and an "All" table that provides general configuration information. Each table can be sorted by clicking the desired column header.

Each item listed in the tables is associated with one of the following severities.

- **INFO.** Successful result.
- **WARNING.** Potential, but not fatal, problem that can negatively impact performance.
- **ERROR.** Issues that prevent Scrittura from running properly.

To use the Check Config feature for custom configuration classes, use the `checkConfigLog ()` method of the `com.ipicorp.tools.config.util.ConfigUtils` class in the `loadConfig()` method of the custom configuration class:

```
void checkConfigLog (CHECK_CONFIG_MODULE module,

String test, CHECK_CONFIG_LEVEL level,

String result)
```

The `loadconfig()` method has the following parameters.

Attribute	Description

module	Name of the module or table in which the check will display.
test	Name of the test being performed (displays in the first column).
level	Defines the severity of the issue (INFO, WARNING, ERROR)
result	Defines the output text of the result (displays in the result column).

Service Pack Tab

The **Service Pack** tab provides information on the current service pack level.

Service Packs being cumulative (as of the current release), this tab provides the list of fixes and enhancements delivered for each service pack (identifier and description).

It also contains an Export button which triggers the export of this information as well as important build and runtime information (such as, environment variables and key Scrittura configuration points) into a Microsoft Excel file.

This file should be provided to Micro Focus Support whenever an issue is encountered and raised with Micro Focus Support.

SetConfig Process

The Scrittura **SetConfig** process allows configuration files to be reloaded into the database from disk, updating all internal processes to use the latest versions. Scrittura will load all configurations from the database during startup.

The SetConfig process also provides the capability to include custom configuration files in the process, or to disable some of the Scrittura core modules not required.

If any required configuration is missing or incorrect, the Administration Console will display a message on an error screen detailing the reason, and provide a button for reloading the configuration.

SetConfig Process Configuration

The SetConfig process is configured in the startup-config.xml configuration file, located in the Scrittura configuration folder. It handles generic application configurations that affect application startup. It also allows the configuration of the SetConfig process.

This configuration file is atypical in that, in addition to being reloaded upon SetConfig, it is always reloaded from disk on startup.

The following configuration sections are included in `startup-config.xml`, each defined within a node located under the root node, `<startup- properties>.`

- <admin-credentials> Node, on the next page
- <password-encryption> Node, on the next page
- <cluster-config> Node, on page 375
- <startup-options> Node, on page 375

- <modules> Node, on page 376
- <custom-modules> Node, on page 377

<admin-credentials> Node

The `<admin-credentials>` node specifies the Scrittura runtime user credentials. This node has two child nodes, `<username>` and `<password>`, that define the credentials of the runtime user. As with all other Scrittura passwords, the password can be encrypted.

<password-encryption> Node

Encryption of all passwords in Scrittura configuration files is handled by the

`<password-encryption>` node. For more details about password encryption, see Password Encryption, on page 379.

The `<password-encryption>` node has the following attribute.

Attribute	Required/ Optional	Description
enabled	Required	Specifies whether passwords for all core Scrittura configuration files are encrypted. Set to true to encrypt all configuration file passwords. Set to false to not encrypt any configuration file passwords. **NOTE:** `docmgr-external-config.xml` is a custom file and is therefor not included in the list of core configuration files.

It is also possible to specify custom files to undergo password encryption. This is done by adding `<file-to-encrypt>` child nodes. Each `<file-to- encrypt>` node must include the following attributes.

Attribute	Required/ Optional	Description
name or absolute-path	Required	Use one or the other to locate the file to encrypt. The `absolute-path` attribute contains the absolute path of the file, while the name attribute contains the name of the file only. If the name attribute is used, the file is assumed to be within the Scrittura configuration folder.
type	Required	Defines the file type, either `XML` or `properties` files

The `<file-to-encrypt>` node must contain at least one `<key>` child node. This `<key>` consists of a pattern by which to locate the password to be encrypted. The syntax of this pattern is determined by the value of the type attribute.

- If the type attribute value is XML, the key should take the form of an XPath expression to locate the node or attribute value to be encrypted. If the XML file in question contains a default

namespace, a custom prefix needs to be defined using the default-namespace node, and used within the XPath expression for any references to nodes within that namespace.

- If the type attribute is properties, the <key> should be a key of the file.

Example

```
<password-encryption enabled="true">
<file-to-encrypt name="docmgr-external-config.xml"
type="xml">
<default-namespace>s<default-namespace/>
<key>//s:bean/s:property[@name='pwd1']/@value</key>
<key>//s:bean/s:property[@name='pwd2']/@value</key>
</file-to-encrypt>
</password-encryption>
```

<cluster-config> Node

The `<cluster-config>` node specifies whether the Scrittura application runs in a cluster. To deploy Scrittura in a cluster, set `its is-clustered` attribute to true.

> **NOTE:** The is-clustered attribute used to be available as the top level is-clustered attribute in scrittura3-config.xml in versions of Scrittura up to 4.3.1 SP1.

<startup-options> Node

The `<startup-options>` node specifies the state of the Scrittura workflow engines upon startup (halted or running), as well as the allocation mode for the different counters.

The `<startup-options>` node includes the following child nodes.

Child Node	Required/ Optional	Description
`workflow-halted-on- startup`	Required	Specifies whether the workflow should be running when Scrittura is started.
`scrittura-halted-on- startup`	Required	Specifies whether message consumption should be running on start-up.
`counters`	Required	Specifies the settings for the different Scrittura counters. Its attributes define how the query should be retried when failing upon reserving a counter range: `retries` is the number of retries and `retry-wait` is the time to wait between two attempts. Its child nodes are counter nodes corresponding to the

		different Scrittura counter. A counter node takes name and block-size as attributes, respectively for the counter name and the range of ID to reserve at once. For more information about counters, see Scrittura Counters, on page 379.

<modules> Node

The `<modules>` node lists the core Scrittura modules. Each module can be enabled or disabled at startup. If a module is enabled, it is expected that all required configuration files be supplied and all necessary dependencies satisfied.

The `<module>` node has the following attributes.

Attribute	Required/ Optional	Description
name	Required	Defines the unique name that identifies the module.
enabled	Required	Specifies whether the module should be enabled at startup. Modules are disabled by default.

The following are valid module names.

Module	Description
birt	BIRT reporting
blogic	Scrittura business engine, BLogic
datamapping	Static Data Framework
docgen	DOCX template based on document generation using the Document Generation Suite (DGS).
economic-panels	Economic panel framework for trade detail screens.
email-dispatch	Email dispatch capabilities.
message-transform	Electronic messaging module (DTCC, ICE, or SWIFT)
queues	User interface configuration capabilities (bulk screens, queue-style search, panel-based trade screens, and so on)
sequencer	Scrittura sequencer
simulate-trade	Trade simulation.
structured-products	Structured Product handling

Some modules, such as BLogic, may contain further configuration if they involve loading files or folders whose storage parameters are configurable. The syntax of the optional file and folder parameters is the same as that for the<custom-modules> Node, below.

<custom-modules> Node

Implementation-specific modules can also be defined that may involve the storage of files and folders within DocManager. These modules are defined as `<custom-module>` child nodes of the `<custom-modules>` node. Declaring the custom modules in startup-config.xml allows them to be included into the SetConfig process, where they are reloaded from disk whenever this process is run.

A `<custom-module>` node has the same mandatory name and enabled attributes as a `<module>` node, and has the following child nodes.

Child Node	Required/ Optional	Description
file or folder	Required	One file child node can be specified and is used to specify the location of its configuration file. One folder child node can be specified in case a folder is associated with the module that needs to be loaded into DocManager.
config-class	Optional	The class attribute of this node defines the Java class to handle the loading of the module configuration. The class must extend the `com.ipicorp.tools.config.Config` abstract class, which provides a number of methods for retrieving the file/folder content. For more information, see the Javadoc documentation of the class.

The `<file>` node has the following child nodes.

Child Node	Required/ Optional	Description

`absolute-path`
or
`relative-path`
	Required	Use one or the other to locate the file on disk. The relative-path is relative to the Scrittura live folder.

`docmgr-location`	Optional	Full destination path to the file in DocManager; this child node is optional and defaults to `/Configuration/`[file name on disk without extension]. All files to be stored under the library root must start with `Library Root/` and all rootless paths should start with `/`.
`entity-type-path`	Optional	Defines the entity type of all parent folders at the desired DocManager destination; this child node is optional and defaults to `Configuration`.

The `< folder>` node has the following child nodes.

Child Node	Required/ Optional	Description

`absolute-path`
or
`relative-path`
	Required	Use one or the other to locate the folder on disk. The relative-path is relative to the Scrittura live folder.

`docmgr-location`	Required	Full destination path to the folder in DocManager; this child node is required and defaults to `/Configuration/[file name on disk without extension]`. All folders to be stored under the library root must start with Library Root/ and all rootless paths should start with `/`.
`entity-type-path`	Required	Defines the entity type of all parent folders at the desired DocManager destination; this child node is required.
`filter`	Optional	Filters the name of the files to be included into the custom module, in case a folder is associated. The following options are available: `Regex`: the body of the filter node may contain a case-insensitive regular expression. `Java class`: the class attribute of the filter node may contain a Java class that handles the filter logic.
`folder`	Optional	This is a recursive element with all the same child nodes as its parent. Nodes relative-path, docmgr- location, and `entity-type-path` are all paths relative to the parent folder.

Example 1: Custom module using a configuration file

```
<custom-modules>
<custom-module enabled="true" name="aCustomModule">
<file>
<relative-path>/someConfigFile.xml</relative-path>
</file>
<config-class class="some.package.SomeConfigClass"/>
</custom-module>
</custom-modules>
```

Example 2: Custom module using a folder

```
<custom-modules>
<custom-module enabled="true" name="aCustomModule">
<folder>
```

```
<relative-path>/aModule</relative-path>
<docmgr-location> Library Root/aModule
</docmgr-location>
<entity-type-path>anEntity</entity-type-path>
<folder>
<relative-path>/aSubModule</relative-path>
<docmgr-location>subModule</docmgr-location>
<entity-type-path>aSubEntity</entity-type-path>
</folder>
</folder>
</custom-module
</module>
```

To add a custom module, declare it under the `<custom-modules>` node. The module should also be assigned a custom class if loading the configuration in memory upon SetConfig is required. This class that should extend the Config class located in the `com.ipicorp.tools.config` package.

Custom modules can be added to the CheckConfig feature, which allows checking whether loading the configuration with the SetConfig process was successful. To learn more about how this can be achieved, see Check Config Tab, on page 372.

Password Encryption

All passwords can be AES-encrypted in Scrittura. Encryption is performed using the startup-config.xml configuration file.

If encryption is enabled, as described in <password-encryption> Node, on page 374, all passwords specified under the password-encryption node of the `startup-config.xml` file will be encrypted.

To encrypt the password, you must specify the encryption key as a 16 character string in a separate file, `encryption-key.ky`, which is located in the Scrittura configuration folder.

Passwords should be specified in clear text before launching the application for the first time. If password encryption is enabled, Scrittura will automatically encrypt the password using the encryption key specified in `encryption- key.ky` when Scrittura is started and the configuration reloaded.

> **NOTE:** Upon encryption, configuration files will be amended to contain the encrypted passwords.

If required, passwords can remain unencrypted by disabling the encryption in `startup-config.xml.`

Scrittura Counters

Scrittura maintains counters for various reasons, including creating the primary key of its various EJBs or for internal use. Counters are stored in the database table IPITOOLS_COUNTER. Scrittura core counters are as follows.

Counter	Description
AUDIT_ANNOTATION AUDIT_CUSTOM AUDIT_DOCMGR AUDIT_VARCHANGE AUDIT_WORKFLOW ipitools_audit	Audit tables primary key counters.
annotations	SCRITTURA_ANNOTATION table primary key counter
messageTicket	SCRITTURA_MSGTKTS table primary key counter
prodinst	SCRITTURA_PRODINST table primary key counter
resources	DOCMGR_RESOURCE table primary key counter
resourceFileText	DOCMGR_RESOURCEFILETEXT table primary key counter
variableName	SCRITTURA_VARNAME table primary key counter
versionctrl_resource	VERSIONCTRL_RESOURCE table primary key counter
WF_RAW_STATISTICS	WF_RAW_STATISTICS table primary key counter

By default, a query is made to the database every time a counter value is required. In order to improve performance, it is recommended to reserve counters by range, particularly under high load. This can be configured in `startup-config.xml`, as described in <startup-options> Node, on page 375. The value of the range should be tuned accordingly to the volume expected by the system.

Any number of counters can also be created for custom use in the application. A helper class, CounterTool, located in the com.ipicorp.tools.remote package is available to access the counters. Counters can be created beforehand in the `IPITOOLS_COUNTER` database table; however the first access to CounterTool will create the counter if the latter does not exist.

Server Hostname, Port, and Protocol

Scrittura makes internal HTTP/S calls to itself to run the signature applet or to check its JNDI setup, among other reasons.

For those calls, the server hostname is set to localhost and the protocol is set to HTTP by default. This however can be a limitation for some Scrittura implementations where, for example, the HTTP port is not activated or `localhost` cannot be used.

It is possible to specify different values for the server hostname, port, and protocol used by the application. The recommended way to proceed is to specify those using the following startup options in the server command line.

Startup Option	Description

`scrittura.hostname`	Server hostname
`scrittura.port`	Server HTTP port
`scrittura.sslport`	Server HTTPS port
`scrittura.enforcessl`	Boolean value that is set to true in order to use SSL.

If `scrittura.hostname`, `scrittura.port`, and `scrittura.sslport` are not specified as startup options, the values for server hostname, HTTP and HTTPS ports will default to the settings made in scrittura-config.xml (respectively `host-name`, `local-port`, and `ssl-port` attributes of the root node, `<scrittura-config>`).

> **NOTE:** The application port (HTTP and/or HTTPS) must always be specified either as a startup option or in scrittura- config.xml.

Startup Options and Custom Properties

Startup options can be defined for Scrittura and added to the startup command using the -D prefix, concatenated to the option name (no space character).

For example, using the scrittura.hostname startup option, and to specify the Scrittura hostname as being `scrittura.bank1.com`, add the following to the Scrittura startup command:

```
-Dscrittura.hostname=scrittura.banka.com
```

Startup options are system properties and can be retrieved using the System class of the J2SE `java.lang` package.

Scrittura also allows the definition of custom properties, not to overload the startup command. Custom properties are defined in the properties file `custom.properties`, directly located under the live repository. Custom properties are name/value pairs and can be retrieved using the `CustomProperties` class of the `com.ipicorp.scrittura.util` package.

For example, add the following line in custom.properties in order to specify the property `maxNumberOfDays` with a value of `10`:

```
maxNumberOfDays=10
```

> **NOTE:** Any change to startup or custom properties requires a server restart. Those cannot be reloaded using the `SetConfig` process.

Run the SetConfig Process

The SetConfig process can be run from the **Start/Stop Scrittura and the Workflow** panel on the General tab of the administration console by clicking the **Reload Configuration** button.

When you log in as the administrative user for the first time, a single screen displays that lets you reload the configuration. If logged on with a non- administrative user, a warning page displays notifying you that the configuration needs to be reloaded by the administrator and no further action is allowed.

A similar message displays if the configuration loaded in Scrittura is incorrect or has errors.

After running the SetConfig process, the FA Tables need to be created (upon initial administrative log in) or recreated and repopulated if any change has been made to queue columns, search columns, or product definitions. This can be performed from the Database Operations tab of the Administration Console. For more information about FA tables, see Fast Access Tables, below

Fast Access Tables

The topics in this section provide an overview of Fast Access tables (FA tables) and provide guidance on tuning.

Fast Access Tables Overview

PI data being stored in a BLOB, the purpose of the FA Tables is to allow a much a more efficient access to frequently accessed PI variables without the need of deserializing the BLOB. Data is also indexed in FA Tables for ease of access and reporting.

Product variables are split into the following categories:

- those that must be accessed efficiently
- those that do not need to be accessed efficiently

Variables used in role conditions, searches, queue displays, and inbound matching dictionaries are stored in a way that minimizes both database and application processing. These variables are referred to as search variables going forward. Other variables need not necessarily be efficiently accessed.

All PI variables are stored as a serialized Hashmap, itself stored in a BLOB column of the `SCRITTURA_VARS` table.

All search variables can also be stored in one or more FA Tables. For maximum efficiency, all variables should fit into a single table since queries for search variables would not need to join other tables.

By storing all variables in a BLOB, initialization of the PI object by the workflow is efficient—a single database request and deserialization. By storing search variables in FA tables, all searches are fast since all variables are in a single table (or small number of tables), requiring few joins.

FA tables span multiple columns. The primary key is the Product Instance PIID while each subsequent column holds a variable value. The column ID is derived from the variable name.

Configure the Fast Access Tables

A top-level attribute, record-size, controls how the FA tables are defined and is configured in `scrittura-config.xml`. `record-size` defines how many bytes can fit into a row of an FA table. Scrittura defaults to a size that matches the stock configuration for the chosen database (1908 bytes for Sybase, 8060 bytes for SQL Server, 65535 bytes for Oracle). If a non- standard database setup exists (such as larger page sizes), this attribute can be used to inform Scrittura to take advantage of the extra row size.

Scrittura uses the declared maximum size (in the VariableValidation element) of each search variable to define the column width of that variable in the FA table. For optimal performance, it is important to minimize the number and size of the FA tables, so take care in defining reasonable

maximum sizes for each String or `OneOf variable`. Variables of type `Date, Integer, CurrencyAmount, Double,` and `Boolean` are treated differently by the system. Column widths for these types track the native database types. If no maximum length is specified for the variable, a default size of 50 characters is used. This default can be controlled by the top-level fa-default-length attribute in `scrittura-config.xml`.

The size for the variable names is also configurable using the top level `attribute fa-table-column-size` in the `scrittura-config.xml` file, the default value being 30 characters.

Scrittura automatically adds all variables used in roles, menus, search- columns, and some internal variables. All other variables are stored in the BLOB but are not available for searching. If your system needs to have more variables accessible through the FA tables, you must specify these variables in scrittura-config.xml. To add a variable in the FA table, add an entry in the `<search-column>` element as follows:

```
<column variable="aVariable" hidden="true"/>
```

FA variables must also be defined in their corresponding Product Definition. Their definition must include their FA Table details using the `FATableName` and `FAColumnName` tags of the `<VariableDefinition>` node, respectively to specify the name of the FA Table they will be added to and the name of the column they will appear as. For more information about Product Definitions, see Product Definitions, on page 55.

Trade Simulation

The Trade Simulation module is a development feature natively provided by the Scrittura core platform. It is fully configurable to handle multiple types of products and can be used to perform load tests if required.

The Trade Simulation module allows for the creation of test trades in the system, through a dedicated user interface. This feature can be used during the development phase in order to test the various Scrittura functionalities.

Trade Simulation is based on pre-defined sample messages whose fields can be overridden manually from within the trade simulation user interface. Any number of sample messages can be used; either XML files or tagfile format.

Multiple trades can be created at the same time, and creation timing tuned as needed if load tests have to be carried out.

In case a copy of the generated tests messages is required, for example to derive market operation messages from the original messages, they can be created offline on the filesystem as XML or tagfile files.

The following topics detail how to configure and use the Trade Simulation module.

- Trade Simulation Prerequisites, below
- Trade Simulation Configuration, on the next page
- Trade Simulation User Interface, on page 387

Trade Simulation Prerequisites

In order to have access to the Trade Simulation, users must belong to the developers group.

NOTE: Trade Simulation is intended for the development phase only. Therefore, the developers group should never be created on production systems.

Trade Simulation Configuration

Trade Simulation is configured in trade-simulation-config.xml, located in the Scrittura configuration folder. Its purpose is to configure the trade simulation screen, the types of trades to simulate, and which fields can be customized from within the user interface.

Its root node, `<trade-simulation>`, has the following child nodes.

Child Node	Required/ Optional	Description
`simulation-properties`	Optional	General properties for trade simulation.
`simulation-files`	Required	Available trade simulations, each defined by a `simulation-file` child node.

General Trade Simulation Properties

General simulation properties are defined by the `<simulation- properties>` node, which has the following child nodes.

Child Node	Required/ Optional	Description
`number-of-trades`	Optional	Number of trades to simulate in a single simulation. Default: 1
`feed-frequency`	Optional	Scrittura feeding frequency, which defines the delay in seconds between two messages, when multiple trades are generated. Default: 1 second
`offline-folder`	Optional	Absolute path of the folder where offline files are generated. Default: empty
`counter-char`	Optional	Character used as a counter and therefore replaced by its numeric value when a trade message is generated. This character can be used in any field. Possible values are limited to: % # ~ _ ^ Default: %

NOTE: With the exception of counter-char, all of these child nodes hold the default values that display on the trade simulation screen. The values can be amended when accessing that screen.

Simulation Files

There can be any number of `<simulation-file>` child nodes under `<simulation-files>`, each of them being based on one pre-defined sample message and corresponding to one of the possible trade simulation available in this module.

A `<simulation-file>` node has the following attributes.

Attribute	Required/ Optional	Description
name	Required	Name of the base message file used for this trade simulation. Files are located under the tagfiles directory, as defined in `scrittura-config.xml`.
type	Required	The type of sample file used for this simulation. Possible values: XML (XML-based files) or TAGFILE (text tagfiles)
is-default	Optional	Specifies whether this simulation displays as the default choice in the trade simulation user interface.
description	Optional	Short description of the trade simulation.

The child nodes of `<simulation-file>` define the fields to override in the base trade file from the trade simulation user interface. The `<simulation- file>` node has the following child nodes.

Child Node	Required/ Optional	Description
`trade-id`	Required	Defines the field to use as a trade reference. It should typically contain a counter character in its default value (such as, ID%) to ensure it is incremented every time a trade is generated.
`confirmation-group-type`	Optional	Defines the field that contains the confirmation group type used for DOCX document generation with DGS.
`template-override`	Optional	If required, defines the field to override the templates to use for the trade when using DGS.
`field`	Optional	Defines a field to override in the base message file. Multiple field nodes can be defined and any field can be overridden.

Each of the `<simulation-file>` child nodes—`trade-id, confirmation- group-type, template-override, field`—have the following attributes.

Attribute	Required/ Optional	Description

label	Required	Label for the field that displays in the trade simulation user interface.
ref	Required	Field to be overridden in the base message. For a tagfile, it is the variable name to override. For an XML message, it is the relative XPath expression for the tag to override. Only the first matching node will be affected.
default	Optional	Default value that displays in the trade simulation user interface, which may contain the counter character defined in the general properties.
values	Optional	Comma-delimited list of possible values for this field. This is an optional attribute and only valid for the field node.

Example

An XML base message (`FXO.xml`, specified in the name attribute of the `simulation-file` node), where two fields can be overridden in the original XML file (Counterparty and Entity Short Code) in addition to Trade ID, Confirmation Group Type and Template Name fields.

```
<simulation-file name="FXO.xml"
type="XML"
is-default="true" description="FXO simulation">
<trade-id label="Trade ID"
ref="tradeID" default="TRD%"/>
<confirmation-group-type label="Confirmation Group Type"
ref="confirmationGroupType" default="FXO"/>
<template-override ref="templateName"
default="MainFXO"/>
<field label="Counterparty Short Code" ref="counterparty/shortCode" default="BNKA"/>
<field label="Entity Short Code" ref="entity/shortCode" default="BNKB"/>
</simulation-file>
```

The corresponding XML section in `FXO.xml` would be:

```
<FXTrade>
<header>
<tradeID>ID001</tradeID>
<confirmationGroupType>FXO</confirmationGroupType>
<templateName />
…
</header>
```

```
<contact>
<counterparty>
<shortCode>UNPTR</shortCode>
…
</counterparty>
<entity>
<shortCode>ABCD</shortCode>
…
</party>
</FXTrade>
```

> **NOTE:** In this example, both tag names for counterparty and entity codes are the same (shortCode). They are differentiated in the configuration by specifying their parent node as well (counterparty or entity).

Trade Simulation User Interface

The Trade Simulation user interface is included in the Scrittura core platform. The corresponding MVC event called for this purpose is 'sim':

`http://server:port/scrittura/controller?e=sim`

The user interface is comprised of the following sections:

- **`Trade Selection.`** Provides options for mandatory information to be included in the simulation, including the base trade file.
- **`Custom Fields.`** Provides options for overriding fields for the simulation.
- **`Trade Input.`** Provides options for how trades will be created and placed in the system.

When trade information and input have been specified, click Create Trades. The trades are then created in the system or offline as specified by the Trade Input section. A confirmation screen will display listing the trades that have just been created.

Trade Selection

The Trade Selection section lets you select which base trade simulation file to use. The additional options available are based on the base file selected.

You must assign a Trade Reference to be used to set the `CommonReferenceID` of the trade in the system once processed.

For trades using the Document Generation Suite (DGS), Confirmation Group Type (and template name when the latter needs to be overridden) can be specified. The Confirmation Group Type option is automatically populated with the list of groups defined in the application.

Custom Fields

The Custom Fields section displays the fields that can be overridden in the simulation. If already set, all fields are pre-populated with the default values set in the configuration. Otherwise, a standard text box is displayed for you to enter (or amend) the value.

Trade Input

The Trade Input section defines how trades are created and placed in the system. You can specify the number of trades you wish to generate and the frequency as a delay in seconds that elapses between two trade generations.

Select the Create files offline check box if you wish to create the corresponding trade simulation files instead of placing trades directly in the system. Those trade files will be created in the specified location.

IT Administration Tasks

This section describes the admin tools available and their use in managing the Scrittura platform. Additional tasks that administrators perform are also explained.

Admin Utility

The Admin Utility allows Scrittura administrators to perform tasks without having to log on to the application itself. Tasks are executed from the command line.

Using the Admin Utility

The Admin Utility main class, com.ipicorp.cmd.scrittura.AdminTool, is included in command-line.jar.

Scripts to run the Admin Utility are provided with the distribution and are located under the `cmdline` folder of the distribution:

- admin.bat for Windows systems
- admin.sh for Unix systems

Using the scripts without argument, the Admin Utility is launched in interactive mode and you can type a series of commands. The Admin Utility can also be launched to perform a specific task by using the scripts with the corresponding command argument.

The following commands are available.

Command	Description
pauseScrittura	Stop message processing
resumeScrittura	Start message processing

stopWorkflow	Stop the workflow engine
startWorkflow	Start the workflow engine
populateFATables [numberOfThreads]	Populate the FA tables
launchCronJob [jobName]	Manually starts a cron job
backFillVariables [numberOfThreads] [timeout] [xmlFile]	Back-fill the database with new variable values
setConfig	Launch the Set Config process
encryptPassword [password]	Outputs encrypted password
help	Prints list of commands
quit	Closes command tool

Connectivity and runtime properties are in the remote.properties file located under the `cmdline/config` directory. The connectivity and runtime properties are as follows.

Property	Description
scrittura.protocol	Protocol to connect to Scrittura (http:// or https://)
scrittura.host scrittura.port	Scrittura server hostname and port number
scrittura.username scrittura.password	Scrittura user credentials
database.url	Database URL. For example, `jdbc:oracle:thin:@localhost:1521:XE`
database.driver	Database fully qualified driver class. For example, `oracle.jdbc.driver.OracleDriver`
database.user database.password	Database user credentials

> **NOTE:** Passwords in the `remote.properties` file can be encrypted upon Scrittura startup/setConfig by adding them to the list of passwords to encrypt in `startup-config.xml`.

Backfill Variables

The variable backfill capability (`backFillVariables` option of the Admin Utility) allows adding variables to existing trades in the database. The command takes the name of an XML file as an argument. The file contains descriptions of the variables to be backfilled:

```
<backfill>
```

```
<variable product="IRS" name="IRS-var3">a-3-value</variable>

<variable product="*" name="A-Common-Var">a-value</variable>

</backfill>
```

A value of "*" for the product attribute means that all PIs are processed, regardless of its product type. Any other value restricts the processing to only PIs whose `ProductDefID` variable matches that value.

If the PI already has a value for the variable in question, nothing is added. If the variable does not exist, it is added with the value specified in the control file.

The `numberOfThreads` option can be used to cause the backfilling to occur on multiple threads.

DocManager Runtime Operations

The following runtime operations can be performed to make alterations to an existing DocManager database:

- Add groups to DocManager
- Change or add indexes after DocManager has been deployed in a production environment
- Add index columns

Add Groups to DocManager

Additional groups can be added to DocManager as needed.

To add groups to DocManager

1. Edit `docmgr-config.xml`, which contains a list of all groups in DocManager.
2. Add the group to the application server security realm.
3. Stop the application server.
4. Run `roleGen.bat`, which adds the new role list to the necessary deployment descriptors for the DocManager API and DocManager Web Application.
5. Start the application server.

Change or Add Indexes Once in Production

After DocManager has been deployed in a production environment, it is possible to alter the Entity Model in order to add or change indexes.

Depending on the change, documents already in the database may need to be updated to reflect the change.

- **Adding an optional index.** No database update required. The index will simply be empty for documents already in the database.
- **Adding a required index.** A database update will need to be made to set a proper value for the index on older documents.

- **Removing an index.** No database update required. The index values for documents already in the database remain in the database.
- **Changing the index type.** The old values will be preserved, but edits or changes to existing documents will require that the index be corrected to the new type.
- **Changing the index label.** The old values will be preserved. You can run a SQL query to clear the old values.
- **Moving (swapping) indexes.** You will need to run a SQL query to update the database.

Add Index Columns

If more than the default number (10) of index columns are desired, then the following process must be adhered to.

1. Set `number-of-indices` in `docmgr-config.xml` to a number bigger than 10.
2. Do a `SetConfig` to ensure that the changes are loaded by the system.
3. Run the `UpdateSchema` event.

DocManager Migration Tool

The DocManager migration tool consists of a command line program that lets you perform document compression. This allows documents which have already been added to the database to be compressed, in batches, according to their extension.

Using the DocManager Migration Tool

Like the Scrittura Admin Utility, the DocManager Migration Tool sends remote requests to Scrittura enterprise beans and as such requires connectivity details to be configured in the properties file, `remote.properties`, located under the `cmdline/config` directory. Properties to configure are identical to the ones required by the Scrittura Admin Utility.

The DocManager Migration Tool main class, `com.ipicorp.cmd.docmgr.MigrationTool`, is included in `command- line.jar`.

Scripts to run the DocManager Migration Tool are provided with the distribution and are located under the cmdline folder of the distribution:

- `docmgrMigrationTool.bat` for Windows systems
- `docmgrMigrationTool.sh` for Unix systems

Using the scripts without argument, the DocManager Migration Tool is launched in interactive mode and you can type a series of commands. The DocManager Migration Tool can also be launched to perform a specific task by using the scripts with the corresponding command argument.

Compress DocManager Documents

The DocManager Migration Tool supports the compression command/argument, which allows the compression of documents in DocManager. `compression` has the following syntax.

```
compression [extension] [compressionLevel] [numDocs] [numberOfThreads] [timeout]
```

where,

- `extension` defines the type of document on which to run the command. Define only one extension and do not prefix the extension with a dot (.), such as "doc". Use the wildcard character "*" to apply compression to all documents regardless of their extension.
- `compressionLevel` is an integer that specifies the degree of compression. Following the standard syntax for java zip compression, the possible values are:
  - -1 for no compression
  - 0-9 for various degrees of compression (9 being the highest level of compression)
- `numDocs` is an integer that defines the number of documents to be compressed in this batch operation.
- `numberOfThreads` is an integer that defines the number of threads that will execute this document compression.
- `timeout` is an integer that defines the amount of time, in seconds, to allow for execution of this compression action.

Roles and Users in Scrittura

This section describes the use and set up of roles and users in Scrittura.

Roles Overview

Roles are defined inside J2EE applications to control a principal's access to the various methods/resources of the application. A principal is a user or group defined inside the application server.

Since these roles can have varying names dependent upon an environment, they are mapped at build time.

A file called 'roles' is placed in subdirectory `top/config/` of the build directory. This is used during the build process to give all the roles relevant permissions in the applications (Scrittura, DocManager, Workflow) through the WEB/ EJB descriptors.

There are two types of declaration of roles inside the Scrittura roles file:

- Each role is listed on its own in the roles file, such as SomeRole.

  Each role must be declared as a security role in a `web.xml` of `scrittura-web.war` to enable that role to be referenced by Scrittura.

  For example, if you have global roles defined with conditions set in the scrittura-config.xml file, these roles also need to be in the roles file to allow Scrittura to reference these roles when checking queue filters/access.

- Each role is mapped, such as

  `SomeRole = ThisPrincipal, OrThisPrincipal.`

  This maps roles to specific principals. For example, the role `ScritturaUsers` may be mapped to the principal `AllScritturaUsers`, meaning that the role inside `Scrittura`

`(ScritturaUsers)` links to the principal (such as a group) `AllScritturaUsers` in the application server.

Scrittura allows anyone in the role ScritturaUsers to log into Scrittura. Therefore, at the J2EE container level, the check to see if a person is in the role `ScritturaUsers` actually means to check if the person is in `AllScritturaUsers`, or is the principal AllScritturaUsers.

For example, in the roles file you might have:

```
ScritturaUsers = AllScritturaUsers admins = mradmin
```

where

- `AllScritturaUsers` is a group defined in the application server with everyone allowed to log into Scrittura.

  This is mapped because it may not be possible to create a group called ScritturaUsers (if they are using LDAP) so the users and groups are retrieved from a directory rather than setup directly in the application server.

- `mradmin` is a user who can perform actions which the admins role is configured to do inside Scrittura, since the client might not want to have a user called 'admin'.

During the build process of Scrittura, the roles listed in the roles file are added into the XML files as the EAR file is built.

Standard and Custom Roles

The following are standard roles for Scrittura:

- admins
- readonly
- editors
- publishers
- reviewers
- ScritturaUsers
- signers_a
- signers_b
- developers
- DMadmins
- DMmanagers
- DMoperators

> **NOTE:** The role ScritturaUsers does not contain a space character between "Scrittura" and "Users" and also contains a capital "S" and "U".

`To add additional roles`

1. Edit the roles file. Each line represents the name of a single role. For example:

   ```
   admins

   signers_a

   signers_b

   ScritturaUsers

   editors

   readonly

   publishers

   Administrators

   TestUsers

   Legal
   ```

   > **NOTE:** When editing the roles file, do not add a line break at the end of the file as this may stop the application server from restarting.

   Save the roles file without an extension. Based on the editor in use, a default file type (such as txt or doc) might be automatically added to the file name. Be sure to eliminate this extension if necessary.

2. Define the group directly in the application server console (or other appropriate location for other J2EE containers).

   This may require administrator access to the container.

3. Stop the J2EE container
4. Redeploy the Scrittura application (with the deploy command).
5. Restart the J2EE container.

Integrate with Other Authentication Systems

In some environments, the J2EE container may be configured to reference a custom authentication domain or other internal system (usually called a `realm`). In such implementations, principals are defined by this separate authentication system and correspond to internal departments generally irrelevant to permissions in Scrittura. These definitions must be mapped to Scrittura roles.

Roles used in permissions may be mapped to multiple externally defined principals. In this case, Scrittura might define a role as:

```
irs-specialist
```

which maps to externally defined principal groups such as:

```
Debt-Doc-Floor-3, Debt-Doc-Floor-4
```

In these cases, the roles file needs to explicitly map each Scrittura role to the externally defined principal, using the following syntax:

```
role[=principal[,principal ...]]
```

and then the roles file might appear as: (excerpt)

```
admins=Administrators

irs-specialist=Debt-Doc-Floor-3,Debt-Doc-Floor-4 ...
```

If an external authentication system defines principal names identical to Scrittura roles, there is no need to add mappings to the roles file in this manner.

Add Users

User definitions and their group memberships are defined directly in the J2EE container using the application server administration console.

User Permissions

In Scrittura, user permissions are handled by an XML file, `scrittura- config.xml`. This file controls the appearance and links available to users when they open an activity worklist. It also can specify conditions where various roles are allowed to open the worklist.

For example, a queue defined as:

```
<queue name="First Review" activity="Scrittura.First Review">

<column display="Counterparty" variable="Party[B]"/>

<column display="Product Type" variable="ProductDefDisplay"/>

<column variable="TradeDate"/>

<column variable="Trader"/>

<column variable="TradeAmended"/>

<view name="Review" type="view" view="/review.jsp"/>

<role name="signers_a" access="write">

<condition>

TradeType='NDF' or TradeType='SWP' or TradeType='FRA'

</condition>

</role>

</queue>
```

sets a `<condition>` such that users with a `signers_a` role can only read and write First Review worklist items where `TradeType` is `NDF, SWP`, or `FRA`.

These conditions must conform to standard `Transact-SQL AND/OR` syntax, using single quotes to begin and end string values.

Example

```
<condition>

(TradeType='NDF' or TradeType='SWP') and Currency='USD'

</condition>
```

If the condition needs to use less-than (<) or greater-than (>) symbols, the entire condition must be enclosed with CDATA tags:

```
<condition>
<![CDATA[Currency='USD' and Notional>5000000.00]]>
</condition>
```

If the condition is not bounded by the symbols:

```
<![CDATA[..........]]
```

then the comparison '<' and '>' symbols will be confused for begin and end XML tags.

Each role with permissions to use a worklist must be specifically listed as a

`<role>` in the `<queue>` definition in `scrittura-config.xml`.

> **IMPORTANT:** Variables used in role conditions should be defined in `commonvars.xml` file and must be given a default value in the Product Definition XML file. If this is not done, any product instances where the value is NULL may fail to show up in queues.

Support for Intersection of Roles in the Queue and Global Role Definition

When defining the access level and condition for a role in the queue definitions in `scrittura-config.xml,` it is possible to provide a list of roles for the role name (such as `name="uk;manager;CDS"`). Only users in those groups will get access. This makes the maintenance of the role definition much easier.

Read and Write Permissions

Access to queues can be controlled in the `<role>` definitions in `scrittura- config.xml` with the access attribute, whose possible values read or write.

For example:

```
<role name="roleName" access="read">
```

If a user is a member of more than one role, they are granted write access in an additive manner (meaning that if they are in any role with write access, they have write access). This read|write functionality is applied after select and on top of any user worklist SQL.

A user with read-only access does not lock the Product Instance or workitem. The user's form will also have only a Close button. Security is checked again on Save. The edit controls are shown, but no changes can be saved.

Permissions on Annotations

Different product types can be configured to display a default set of annotation threads. Each of these threads may be restricted for reading and writing based on role membership. This can be configured with the `<annotations>` element in a Product Definition file or globally in `scrittura-config.xml`.

Determine the List of Visible Work Items

In a given queue, displayed workitems are selected as:

```
the UNION of all role-based conditions on the queue INTERSECT

the UNION of all global role conditions INTERSECT

any custom pseudo-sql conditions requested on the URL used to link to the activity
list (an optional configuration feature whereby links to activity lists defined in
JSP pages can impose additional query-like conditions).
```

From this worklist, based on the user's roles, the items that the user has write access to are determined. This is based on the maximum rights granted by the union of role-based conditions on the queue.

Example: Everyone Can See Their Own Workitems

A typical configuration setting would be to allow each user to read and write "their own" workitems in queues, while members of the "admins" group can read and write all the workitems.

In the following example, the product variable "docSpecialist" has been configured to store the name of a user. It can be assigned using beanshell scripts or BLogic rules as a document passes through the workflow, or manually assigned by another user in an earlier queue.

```
<queue name="First Review" activity="Scrittura.First Review">

<column display="Counterparty" variable="Party[B]"/>

<column display="Product Type" variable="ProductDefDisplay"/>

<column variable="TradeDate"/>

<column variable="Trader"/>

<view name="Review" type="view" view="/review.jsp"/>

<role name="admins" access="write" />

<role name="" access="write">

<condition><![CDATA[docspecialist='USERID']]></condition>

</role>

</queue>
```

In this scenario, two roles are established:

- Anyone in the "admins" role can read and write the document.
- Anyone whose USERID matches the "docspecialist" variable can also read and write the document.

Mutually Exclusive Queues

The built-in variables `UserId, LastEditUser,` and `LastForwardUser` can be used to impose mutually exclusive functionality whereby the same user cannot edit a document and push it forward an approval queue.

This is accomplished by adding to the role definition for an activity in the workflow definition:

```
<role name="" access="write">
<condition>
<![CDATA[LastForwardUser <> 'UserId']]>
</condition>
</role>
or
<role name="" access="write">
<condition>
<![CDATA[LastEditUser <> 'UserId']]>
</condition>
</role>
```

Locks

Scrittura employs optimistic locking, which means that when user A opens a document, other users attempting to open the document are notified that the original user has the trade open. The other users can still make changes, but user A is prevented from making any further changes.

This lock can be set to expire after a certain period of time. Set the `<lock- expiration-secs>` value in `scrittura-config.xml` to set the number of seconds before expiration.

Scrittura Logs

Log files for Scrittura are normally located in the logs subdirectory of the Scrittura home directory. Log configuration is performed in the `log.cfg` file, located in the Scrittura configuration folder.

Log files record all events and errors related to Scrittura. Any error performed by the server itself, outside the context of the application, such as a compiling JSP files, will only go to the server log or can be checked in the WebLogic server console.

Attributes (such as maximum size, file rotation) for the log can also be set in `log.cfg`. For more details regarding the log, see the *ApacheLog4j* documentation.

The application server also maintains its own logs inside its installation directories. The behavior of those log files can generally be configured (such as nightly rolling or truncation of logs). For an overview of the different capabilities, see the application server documentation.

Using standard log4j conventions, it is possible to configure multiple logs to record events of various severities. Severity levels of interest can be set using a category filter. You can also set a maximum size for this file as well as a maximum number of older log files to keep on file.

A common configuration is to send all messages to a single file (in the example, configured at /opt/scrittura/logs/scrittura.log) and to additionally copy messages of a given severity level out to the console window from which the J2EE container was started.

Mail Log Errors to Administrator

The log file can become very large if verbosity is set to debug, and it might be hard to catch errors as they appear. The following is a log4j configuration to set up an appender that will email the administrator upon a predefined condition. It will send an email containing the logs generated just before the error condition too. It can also be modified to send an email to the administrator upon a specific failure (such as a cronjob failure).

```
<appender name="SMTP" class="org.apache.log4j.net.SMTPAppender">
<errorHandler class="com.scrittura.ErrorHandler"/>
<param name="Threshold" value="ERROR"/>
<param name="To" value="..."/>
<param name="From" value="..."/>
<param name="Subject" value=""/>
<param name="SMTPHost" value="..."/>
<param name="BufferSize" value="100"/>
<layout class="org.apache.log4j.PatternLayout">
<param name="ConversionPattern" value="[%d{ABSOLUTE},%c{1}] %m\r\n"/>
</layout>
</appender>
```

where `ErrorHandler` is in the form of:

```
class ErrorHandler extends TriggeringEventEvaluator
{
public boolean isTriggeringEvent(LoggingEvent event)
{
if (event.level.isGreaterOrEqual(Level.WARNING))
{
return true;
}
else if (event.level.isGreaterOrEqual(Level.INFO))
{
return event.getMessage().contains("some condition");
}
return false;
}
```

```
}
```

Display Transaction Information in the Logs

A customized Log4J PatternLayout adds the application server transaction and username information to the log. Use this class instead of `org.apache.log4j`.

Use PatternLayout in your log.cfg to get this additional transaction and user information in the log. Extra info is added in square brackets before any other log information specified by the PatternLayout pattern:

```
log4j.appender.R.layout=com.ipicorp.tools.log4j.TransactionPatternLayout
```

Add a Logging Category to Beanshell

The following BeanShell adds the contents of a message ticket to the log under a newly created category called TagOutput. The first line in the following example would appear in a single line of BeanShell.

```
Category log = Category.getInstance("TagOutput.bsh"); log.debug("Tags : " + tags);
```

Workflow Notifications

The Scrittura workflow supports various types of notification in order to let users or administrators know of important events in the workflow. That can be sending an email to users to let them know that an important step has been completed for a trade (such as, its confirmation has been dispatched) or warn the IT administrators of issues in the workflow (such as, trades in error state or stalled).

Email Notification

Scrittura supports workflow-based email notifications, where a user receives an email as a document passes through a BeanShell or classtool activity.

Workflow Error Notification

Administrators or IT support can receive notifications of workflow errors in Scrittura by configuring the BatchEmailErrorState class in scheduler.xml.

Example

Scan Scrittura for workflow errors every hour on the 30 and 59 minute marks for all business days

```
<job class="com.ipicorp.scrittura.scheduler.jobs
.BatchEmailErrorState"
name="EmailErrors" runAs="admin">
<schedule day="*"
hour="*" min="30,59"
month="*" weekday="1-5"/>
```

```
<param key="emailFrom" value="scrittura@banka.com"/>
<param key="emailTo" value="support@banka.com"/>
<param key="emailCc" value=""/>
<param key="emailBcc" value=""/>
<param key="emailSubject"
value="Scrittura Workflow Error Report"/>
</job>
```

Stalled Item Email Notifications

Administrators or IT support can receive notifications when items get stalled in the Scrittura workflow by configuring the StalledItemsNotifier class in `scheduler.xml`.

The `StalledItemsNotifier` configuration also takes an additional `stalledSeconds` attribute, which represents the least number of seconds an item has to be stalled in order for it to be considered a stalled item. Scrittura recommends at least one hour (3600 seconds) for this attribute.

However, this setting depends on the average load on the Scrittura workflow.

Example

Scan Scrittura for stalled items every hour on the 30 minute mark for all business days.

```
<job class="com.ipicorp.scrittura.scheduler.jobs
.StalledItemsNotifier"
name="FindStalledItems" runAs="admin">
<schedule day="*"
hour="*" min="30" month="*" weekday="1-5"/>
<param key="stalledSeconds" value="3600"/>
<param key="emailFrom" value="scrittura@banka.com"/>
<param key="emailTo" value="support@banka.com"/>
<param key="emailCc" value=""/>
<param key="emailBcc" value=""/>
<param key="emailSubject"
value="Scrittura Stalled Items Report"/>
</job>
```

Example Stalled Items notification email:

The following item(s) have been stalled in the Scrittura workflow for over 3600 seconds in the following locations:

```
Workflow.Activity | Arrival Transition ID | Workitem ID | State | EnqueueTime
1) STALLER_Class | applyAutoSigs | 603 | ready | 2014-03-10 13:38:12.0
```

Performance Tuning

This section defines performance tuning guidelines for Scrittura. The intention is to provide the following benefits:

- Rapid throughput
- Rapid response time
- High scalability
- Elimination of performance related errors

The section highlights possible performance issues together with recommendations to resolve those issues. It also explains possible scalability approaches for Scrittura.

Possible Performance Issues

When configuring Scrittura, performance issues may be encountered. This section discusses some issues that may affect performance; possible solutions are discussed in Performance Recommendations, on the next page.

Low Throughput

The throughput refers to the amount of work completed in a fixed time period. It can vary in different installations and depends on the configuration of a number of key elements, such as Java Virtual machine, Database, application server, and Operating System.

Application Slowdown Mostly During Peak Load

The system can slow down and result in an increased response time for searches, document retrievals, or general browsing of Scrittura application. These symptoms are most prominent during peak load hours in a typical business day.

Application Crash

The application server hangs or runs out of memory, leaving behind a hotspot error log file (such as, `hs_err_pid1884.log`) in the application installation directory of the application server.

Database Blocking

Database blocking can occur when one rogue transaction holds on to a lock on a table while other transactions wait for the release of the lock. This can happen when a search or a report is run from Scrittura that could result in huge ResultSet or table joins. These symptoms have been noticed primarily on FA tables and the `DOCMGR_RESOURCEACCESS` table.

CacheFullException

A `CacheFullException` is thrown by the application server when the entity bean cache is full. This can occur if the configured cache limits are too low or a recent increase in load on Scrittura requires further tuning of the cache size.

Application-level caching is used by default whenever an entity bean does not specify its own cache in the application server descriptor.

Do not try to configure the cache size/bean. It is very difficult to predict the maximum number of entity beans that will be loaded in a given transaction since you would not know what the user is trying to do. If the number of beans that are getting loaded into the memory exceeds the `max-beans-in-cache` setting for a given transaction, the application server will throw a `CacheFullException`. Configure an application level cache instead and specify a large value for `max-beans-in-cache` and test your application under max load. Test with new limits and adjust accordingly. You must have enough memory when you start the application server or you may receive an `OutOfMemoryError`.

OutOfMemoryError

An `OutOfMemoryError` is thrown by the JVM when it runs out of the memory required to allocate space for creating new objects into the memory. These errors can also lead to hotspot errors (look for a hotspot error log file). These errors can be found in the application server console output file as well as in the Scrittura log file.

The most common reasons for seeing an `OutOfMemory` excpetion are as follows:

1. Insufficient memory settings.
2. Improperly configured JVM memory segments, namely "Young generation" "Tenured generation" and "Permanent generation".
3. Improper choice of Garbage collection algorithm, where the pace of garbage collection cannot keep up with memory demands even though there is ample memory available to the JVM process.

Performance Recommendations

For effective performance, this section provides recommendations for the application server, JVM, Application tuning, and JMS tuning. Further information can be located in the vendor documentation for these components.

Application Server Performance Recommendations

The following performance recommendations are related to the application server.

Tune Number of Execute Threads

Care should be taken to configure the number of Execute threads. A very high number can decrease the performance and a very low number can do the same. Remember that an excessive number of execute threads can lead to the unnecessary consumption of resources like memory and CPU cycles and context switching; this may adversely affect the throughput instead of increasing it.

The best way to tune this number is to monitor the throughput during maximum load and adjust the value up until the throughput starts declining. The decline indicates that the context switching is taking its toll. As a general guide, the default (25) is probably not enough, and there seems to be no adverse impact on context switching up to 80 execute threads (remembering of course that in the default configuration 33% are allocated as socket reader threads).

Additionally, optimization for user response can be accomplished by separating the application server execute threads for user/http requests.

Tune JDBC Connection Pool Maximum Capacity

If the JDBC connections used is always equal to maximum capacity, you should tune the number of connections. Target is 70% to 80% utilization at normal load. Usually a request should not be seen waiting for a connection. The `InitialCapacity` and `MaxCapacity` attributes of the `JDBCConnectionPool` element allow you to set the initial and maximum number of physical database connections that a connection pool can contain.

It is advisable that the number of connections in the pool equal the number of concurrent client sessions that require JDBC connections. The pool capacity is independent of the number of execute threads in the server. There may be many more ongoing user sessions than there are execute threads.

Number of Prepared Statements

The number of SQL Prepared Statements should be tuned in the application server console to a value in line with the Scrittura application. A value of 200 can be a starting point, to be refined according to the implementation.

Timeouts

To avoid unnecessary rollbacks, the JTA timeout should be tuned in the application server console to a value in line with the Scrittura application. The JTA timeout must control the maximum duration of a transaction, hence all other timeouts (such as, JDBC connection timeouts) should be set to a higher value.

> **NOTE:** The requeue-secs parameter in workflow.xml should also be set to a higher value.

A value of 180 seconds for the JTA timeout can be a starting point, to be refined according to the implementation. A good value depends on the general time that transactions take to complete in the application.

Set Cache Size for Entity Beans

If the EJB cache size is too small it may cause a `CacheFullException` in the WebLogic Server. The performance may get a boost just by optimizing the EJB cache limits. To monitor the various cache parameters like "cache miss ratio" use the application server console.

If you are running many finders or home methods, or creating many beans, you may want to tune the `max-beans-in-free-pool` element so that there are enough beans available for use in the pool. Use the `max-beans-in- cache` element of the application server descriptor to specify the maximum number of objects of this bean class that are allowed in memory.

Use WebLogic Server "Native IO" Performance Packs

Applies to WebLogic only

This could be considered if I/O operations like uploading a large document take long time to complete. Performance packs use a platform-optimized, native socket multiplexer to improve server performance.

For example, the native socket reader multiplexer threads have their own execute queue and do not borrow threads from the default execute queue, which frees up default execute threads to do application work.

However, if you must use the pure-Java socket reader implementation for host machines, you can still improve the performance of socket communication by configuring the proper number of socket reader threads for each server instance and client machine.

Tuning Fast Access Tables

It is possible to limit the size of the `VCHAR` variables in the FA tables by using the `maxLength` element, thereby reducing the number of FA tables.

When possible, reducing the number of FA Tables also increases performance as smaller joins are necessary for database queries. FA Tables should also be indexed accordingly.

JVM Performance Recommendations

The following performance recommendations are related to JVM.

Sun HotSpot Technology

Ensure that Java has its first option as -server or -client to enable Sun HotSpot technology. This enables JIT compilation that should boost the performance noticeably.

Young Generation Size

Keep the Young generation size approximately 25% of total allocated memory size. The option `-XX:NewSize` enables the configuration of Young generation. In the following example, the size 768 MB is approximately 25% of the total memory size.

Example of memory arguments:

```
-Xms2944M

-Xmx2944M

-XX:NewSize=768M

-XX:MaxNewSize=768M

-XX:MaxPermSize=256m

-XX:+ShowMessageBoxOnError

-XX:+PrintGCDetails

-XX:+PrintGCTimeStamps

-XX:+PrintHeapAtGC

-Xloggc:gc.log

-XX:+UseConcMarkSweepGC

-XX:+UseParNewGC
```

```
-XX:+CMSParallelRemarkEnabled

-XX:+UseDefaultStackSize

-Xss320K"
```

Permanent Size

Consider increasing the Permanent size (the memory area where Java classes are loaded) if Java class loading/unloading is excessive. This can be diagnosed by enabling the `–verbose:gc` option and monitoring the application server stdout log file, or using the `-XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+PrintHeapAtGC -Xloggc:gc.log` options to enable more details and routing all the information to gc.log file. Usually 128MB should be sufficient; anything above 256MB is not going to make much difference.

Example of memory arguments:

```
-Xms2944M

-Xmx2944M

-XX:NewSize=768M

-XX:MaxNewSize=768M

-XX:MaxPermSize=256m

-XX:+ShowMessageBoxOnError

-XX:+PrintGCDetails

-XX:+PrintGCTimeStamps

-XX:+PrintHeapAtGC

-Xloggc:gc.log

-XX:+UseConcMarkSweepGC

-XX:+UseParNewGC

-XX:+CMSParallelRemarkEnabled

-XX:+UseDefaultStackSize

-Xss320K"
```

Garbage Collection

Consider using Parallel Garbage Collection algorithms in combination with concurrent GC for different memory segments. This would help keep the Garbage Collection in pace with memory allocation requirements without significant JVM halts and delays.

```
-XX:+UseConcMarkSweepGC

-XX:+UseParNewGC

-XX:+CMSParallelRemarkEnabled
```

JMS Performance Recommendations

The following performance recommendations are related to JMS.

JMS Server/Connection Factory

You may need to consider the application server JMS server/connection factory tuning. If you turn off the Scrittura message throttling (0 seconds), the JMS runs at full speed and can often cause the application server to run out of memory as the messages get backed up. Instead of artificially throttling using the Scrittura configuration, it is better to configure the Connection Factory to handle the throttling based on various parameters such as number of messages and size.

JMS Redelivery Settings

Re-queuing of messages due to Transaction rollbacks can sometimes cause issues. If messages are being re-queued due to permanent errors, system slowdowns can occur. Redelivery settings should be carefully thought over or reassessed to minimize this situation.

Sometimes an implementation will have a class tool which is very resource intensive. You can set these class tools to run on a single threaded priority so that only one process can run at a time. Another thing to consider in these cases is the redelivery limit, smaller is better. Application programmers should also consider if retrying a transaction is the correct thing to do. For example if a file does not exist (and never will) it is not worth rerunning the transaction.

Scalability

Scrittura can be scaled up using one or more of the following options.

- Deployment architecture
- Environment tuning

Deployment Architecture

Scrittura requires external or third-party servers to run outside the application, for example:

- Database server
- PDF Conversion server

It is recommended to run the Scrittura application and those two servers on separate boxes in order to guarantee their full access to the box resources (such as CPU) and maximize performances.

Similarly when Scrittura runs in a cluster, although multiple nodes can be deployed on the same box, it may be worth deploying them on different boxes if performance issues arise.

General Environment Tuning

Consider the following general environment tuning options if you experience performance issues.

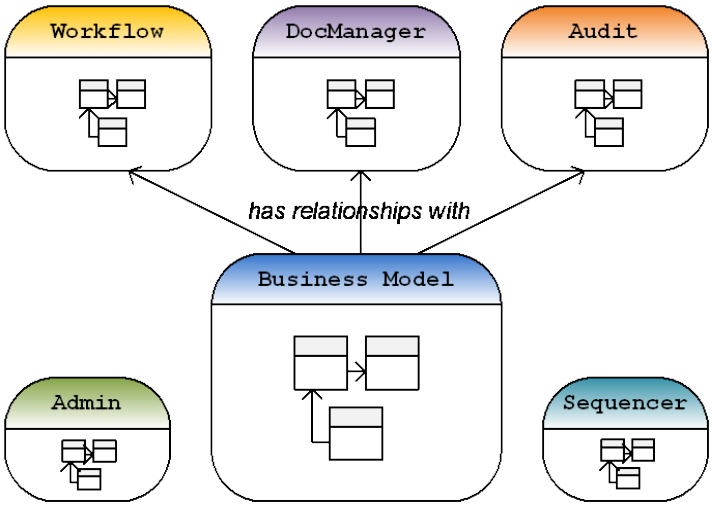
- **Thread Pools.** A low thread pool count can lead to clients waiting for responses; a high count can lead to unnecessary resource utilization, high CPU utilization, and expensive context switching. The target is for CPU utilization remains less than 70% of its full capacity at all times.
- **Database Connection Pools.** Target is 70% to 80% utilization at normal load. Usually a request should not be seen waiting for a connection.
- **Database Cache Size.** Make sure the prepared statement cache in the database is sufficiently large to cache all plans.
- **Garbage Collection.** If the garbage collector cannot free enough memory to hold the new object, it throws an OutOfMemoryError. Using a different GC collection scheme for Tenured and Young Generation (preferably parallel for Young and concurrent for Tenured Generations) should boost the performance.
- **Memory Segment Size.** Ensure that the Perm, Tenured, Young Generation sizes are set as per the Recommendations.
- **Database Optimization.** The database size should be monitored and performance optimized by the removal of expired PI data. Scrittura Archiving or other tools may be used for this purpose. Performance may be improved through additional indexing.

Appendix A: Scrittura Data Model

This section contains the following topics:

Data Model Overview

Data are persisted in the Scrittura database for its different components and can be divided into the following categories.



The following are the different data model categories.

Category	Description
Business Layer	The Business Layer is the core of the data model and contains all data related to trades and their representations in the system.
DocManager	The DocManager data model encompasses all data related to document storage (resources, versions, etc).

Workflow	The workflow tables contain all objects related to trade handling within the Scrittura workflow (activity items, workitems, etc).
Audit	Audit includes all audit records, which are used across the application by its key components (workflow, DocManager, etc).
Administration	Administration tables
Sequencer	Additional tables necessary for the sequencer
Static Data	Additional tables to store all static data within Scrittura. All static data tables can be fully configured and are specific to each implementation.

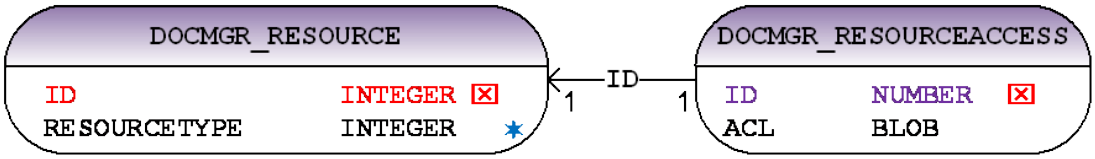
The category illustrations in this section depict the tables belonging to those categories, and highlights the relationships between them.

All illustrations use the following legend and conventions:

- Red. This field belongs to the primary key.
- Blue. This field belongs to a foreign key.
- Purple. This field belongs to both the primary key and a foreign key.
- ☒. This field cannot be null.
- ✶. This field is indexed.
- Types displayed are generic (VARCHAR for strings, NUMBER for doubles, etc).
- The foreign key along with the multiplicity is directly displayed on the relationship.

Example

In excerpts from the two tables `DOCMGR_RESOURCE` and `DOCMGR_RESOURCEACCESS`, the illustrations are interpreted as follows.

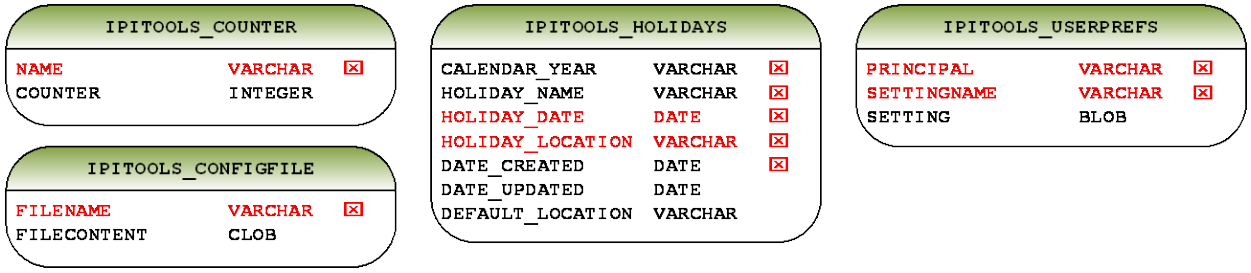


- `DOCMGR_RESOURCE` primary key is ID.
- `DOCMGR_RESOURCE ID` field cannot be null.
- `DOCMGR_RESOURCE RESOURCETYPE` field is indexed.
- `DOCMGR_RESOURCEACCESS` primary key is ID.
- A Foreign Key is defined for `DOCMGR_RESOURCEACCESS: ID`.
- `DOCMGR_RESOURCEACCESS ID` field cannot be null.
- There is a one-to-many relationship between `DOCMGR_RESOURCEACCESS` and `DOCMGR_RESOURCE`.

> **NOTE:** Although relationships and foreign keys are indicated on this diagram these are not database managed and are included for illustrative purposes only.

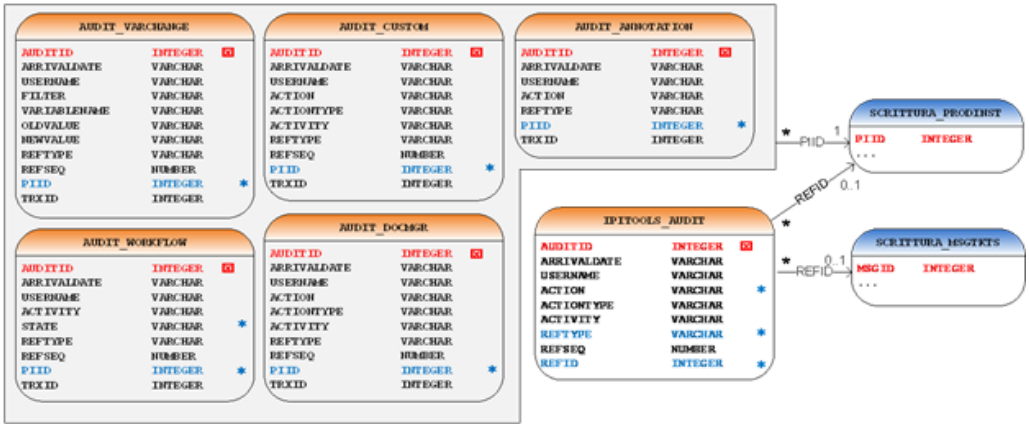
Data Model: Administration Category

Diagrams in this section detail the data model for the Administration category.



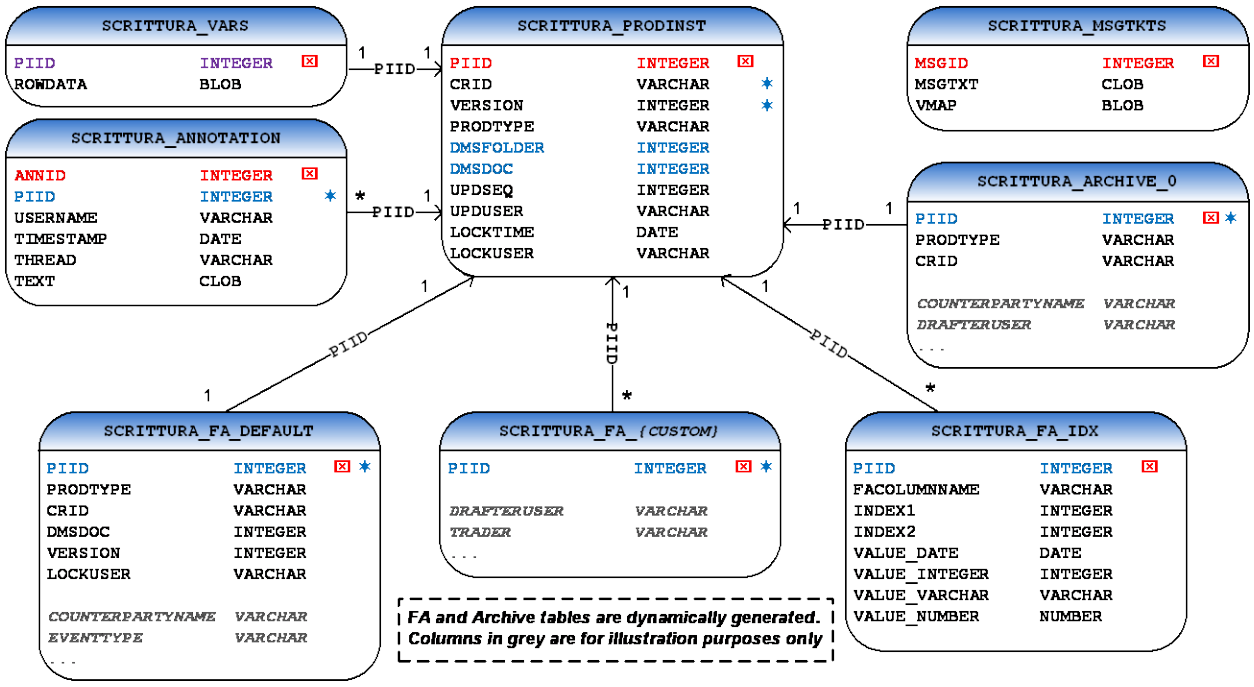
Data Model: Audit Category

Diagrams in this section detail the data model for the Audit category.



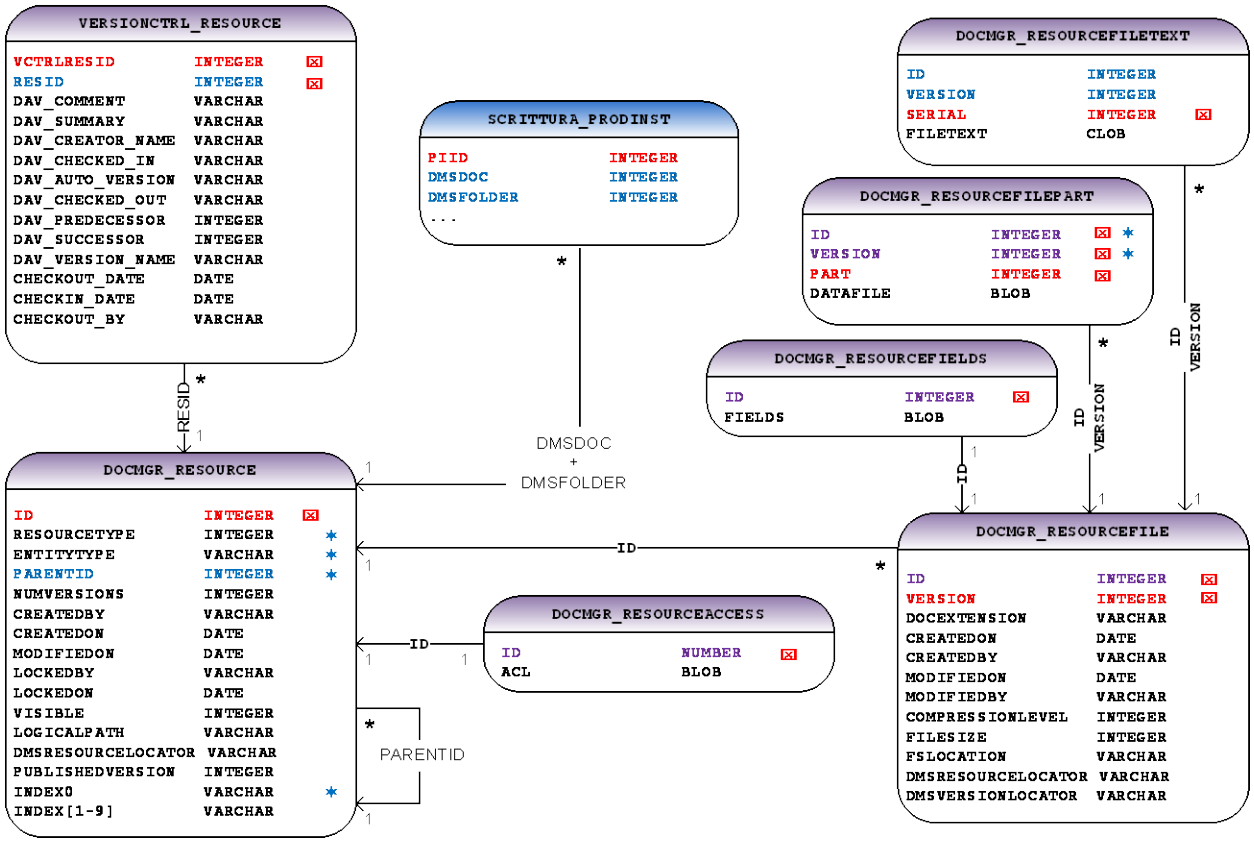
Data Model: Business Model Category

Diagrams in this section detail the data model for the Business Model category.



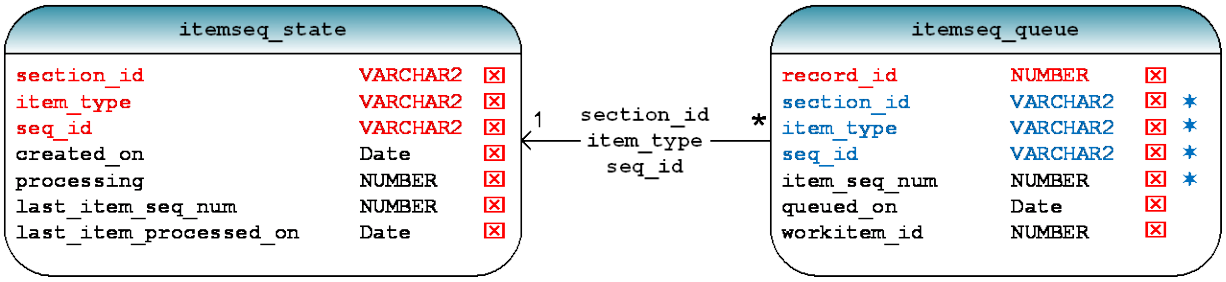
Data Model: DocManager Category

Diagrams in this section detail the data model for the DocManager category.



Data Model: Sequencer Category

Diagrams in this section detail the data model for the Sequence category.

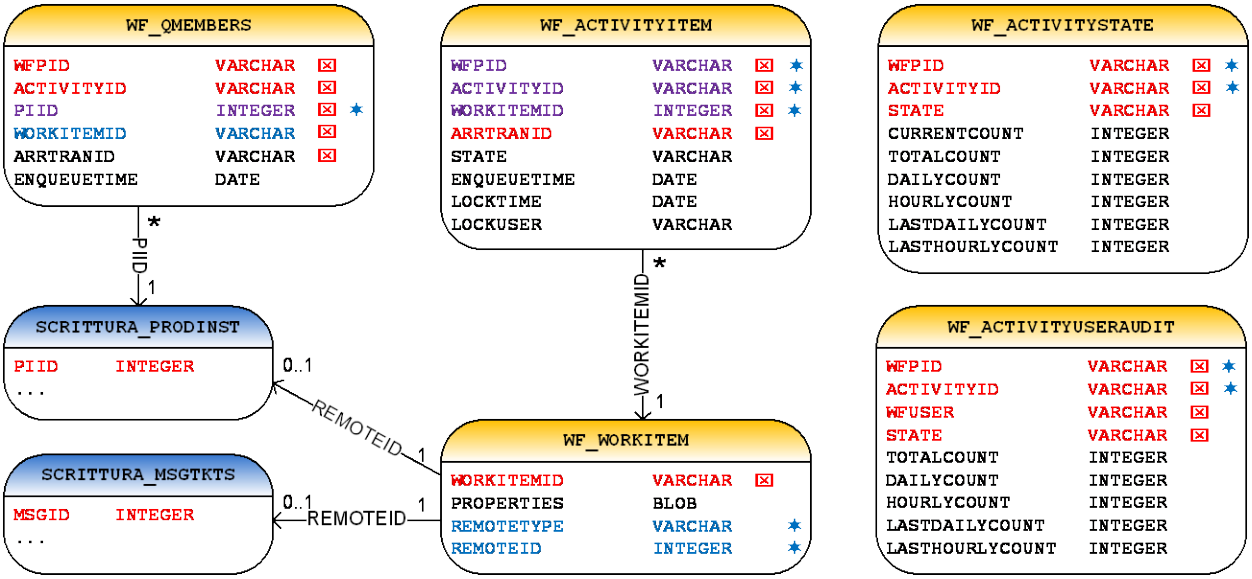


Data Model: Static Data Category

Static Data tables are dynamic tables whose content is fully configurable based on your needs. Once the system is deployed, the DDL (Data Definition Language) is automatically generated allowing the creation of the Static Data tables in the Scrittura database.

Data Model: Workflow Category

Diagrams in this section detail the data model for the Workflow category.



Appendix B :Sample Trade Detail and Bulk Screen Panels

This section provides details and examples on how to write custom panels and process handlers for trade detail screens and bulk screens.

This section contains the following topics:

Bulk Panel Sample

A bulk panel is technically a fragment of JSP that will be included into a table, itself included into a form. Hence it just contains the table rows you wish to display in screen.

Example

```
<%
String pkey = (String)request.getAttribute("pkey");
%>
<tr>
<td colspan="2" height="10" />
</tr>
<tr>
<td>Unlink components from Structure:</td>
<td>
<input type="hidden"
name="SaveAndClose" value="Save & Fwd" />
<input type="button"
value="Unlink" onclick="validateSubmit(this.form, '<%=pkey%>',
'Save & Fwd');"

</td>
```

```
</tr>


class="btn"  onmouseover="this.className='btn
btnhov'"onmouseout="this.className='btn'" />


<tr>
<td colspan="2" height="10" />
</tr>
```

To submit the form (and trigger the execution of the panel's process handler), add a control of type button that calls the javascript validateSubmit() with the following parameters:

- Current form
- Request's pKey attribute
- Action (e.g. 'Save & Fwd', 'Save & Close')

Bulk Trade Handler Sample

Bulk Trade Handlers extend the class BaseBulkTradeHandler located in the package `com.iwov.gcm.scrittura.web.queue.`

BaseBulkTradeHandler's API is as follows. Methods `validateBulk(), executeBulkProcessing(),` and `executeTradeSpecificProcessing()` can be overridden.

```
public static final int STOP_PROCESSING = 0 ;
public static final int CONTINUE_PROCESSING = 1 ;
/**

*       Process handler validation to validate the bulk selection
*       against the intended action.
*       No validation is performed by default; override this method
*       to implement custom validation.
*       Note this method should not amend trade data.
*
*       @param itemIds  Array of ActivityItemID to process
*       @param sc       Servlet ccontext
*       @param request  HTTP request
*       @param response HTTP response
*       @return true if validation is successful, false otherwise
```

```
*/

public boolean validateBulk(String[] itemIds,
ServletContext sc, HttpServletRequest request, HttpServletResponse response)

{
return true;
}
/**

*       Execute specific or additional bulk processing for a
*       certain event
*
*       @param itemIds  Array of ActivityItemID to process
*       @param sc       Servlet ccontext
*       @param request  HTTP request
*       @param response HTTP response
*       @return STOP_PROCESSING or CONTINUE_PROCESSING
*       depending whether the global event processing   must stop
*       or continue after this
*/

public int executeBulkProcessing(String[] itemIds,
ServletContext sc, HttpServletRequest request, HttpServletResponse response)
throws Exception
{
return CONTINUE_PROCESSING;
/**
}
*       Execute trade-specific processing for a certain event
*
*       @param itemId   Trade ActivityItemID to process
*       @param sc       Servlet ccontext
```

```
*       @param request  HTTP request
*       @param response HTTP response
*
*/


public void executeTradeSpecificProcessing
(String itemId, ServletContext sc, HttpServletRequest request)
throws Exception
{
return;
}
```

Trade Detail Panel Sample

A trade detail panel is technically a fragment of JSP that will be included into the global trade detail JSP. Unlike bulk panels, trade detail panels contain their own layout and form (if required).

Example

```
<%@ taglib uri="scrittura3.tld" prefix="scrittura"%>
<table width="100%" border=0 cellpadding=0 cellspacing=0>
<table width="100%" cellpadding="0" cellspacing="0">
<!-- Panel header -->
<tr class="tabletop0">
<td height="20px" class="topBorder">
<a href="javascript:maximiseWidget('dPanel');" style="text-decoration: none;">
<img src="images/maximise.gif" id="maximiseButtondPanel" style="border-width: 0px;
vertical-align: middle; display: none" />
</a>
<a href="javascript:minimiseWidget('dPanel');" style="text-decoration: none;">
<img src="images/minimise.gif" id="minimiseButtondPanel" style="border-width: 0px;
vertical-align: middle; display: inline" />
</a>
 Panel Header
</td>
</tr>
```

```
<!-- Panel content -->
<tr id="dPanel" name="dPanel" class="label-text">
<td>Panel content</td>
</tr>
</table>
</table>
```

Single Trade Handler Sample

Single Trade Handlers extend the class `BaseSingleTradeHandler` located in the package `com.iwov.gcm.scrittura.web.queue.`

`BaseSingleTradeHandler's` API is as follows, and the `executeSingleTradeProcessing()` method can be overridden.

```
/**

*       Execute specific or additional processing for a certain
*       event applying to a single trade
*
*       @param ins      The Product Instance
*       @param map      PI variable map
*       @param queueRoute       Queue route variable value
*       @param request  HTTP request
*/

public void executeSingleTradeProcessing
(ProductInstanceLocal ins, HashMap map,
String queueRoute, HttpServletRequest request)
throws Exception
{
return ;
}
```

Appendix C: Configuration Files

This section contains the following topics:

Sample: docmgr-config.xml

```
<docmgr-config use-entity-types="true"
security-factory="com.ipicorp.docmgr.security
.PerResourceAclFactory"
disable-security="false" superuser="system" database="mssql">
<field-type type-name="String"
validation-class="com.ipicorp.docmgr.validation
.StringValidator" />
<field-type type-name="Integer"
validation-class="com.ipicorp.docmgr.validation
.IntegerValidator" />
<field-type type-name="SSN"
validation-class="com.ipicorp.docmgr.validation
.SSNValidator" />
<field-type type-name="Zipcode"
validation-class="com.ipicorp.docmgr.validation
.ZipcodeValidator" />
<field-type type-name="Year"
validation-class="com.ipicorp.docmgr.validation
.YearValidator" />
<field-type type-name="Quarter"
validation-class="com.ipicorp.docmgr.validation
.QuarterValidator" />
<field-type type-name="Date"
validation-class="com.ipicorp.docmgr.validation
```

```
.DateValidator" />
<import-dir dir="C:\import\monitor_queue_1" use-xml-metafiles="true"
sleep-seconds="10" />
<import-dir dir="C:\import\monitor_queue_2" use-xml-metafiles="false" sleep-
seconds="10" />
<configured-roles>
<role>readonly</role>
<role>editors</role>
<role>publishers</role>
<role>admins</role>
</configured-roles>
<fax>
<fax-enabled>true</fax-enabled>
<fax-class> com.ipicorp.docmgr.banka.BankADocmgrFaxer
</fax-class>
</fax>
</docmgr-config>
```

Sample: entity-types.xml

```
<dms-entity-model name="TestModel">
<entity-acl>
<read-list>
<group>admins</group>
<group>publishers</group>
<group>editors</group>
<group>readonly</group>
</read-list>
<write-list>
<group>admins</group>
<group>publishers</group>
</write-list>
<write-doc-list>
<group>admins</group>
```

```
<group>publishers</group>
<group>editors</group>
</write-doc-list>
<create-list>
<group>admins</group>
<group>publishers</group>
</create-list>
<del-list>
<group>admins</group>
<group>publishers</group>
</del-list>
<security-list>
<group>admins</group>
</security-list>
</entity-acl>
<entity-type name="Counterparty"
title-index="1" doc-title-index="9"
inherit-parent="false">
<index idx="0" label="Title" index-type="folder"
required="true" type="String"/>
<index idx="1" label="Counterparty" index-type="folder"
required="true" type="String"/>
<index idx="8" label="Doc Date" index-type="document"
required="false" type="Date"/>
<index idx="9" label="DocType" index-type="document"
required="true" type="String"/>
<field idx="0" label="Address" field-type="folder"
required="false" type="String"/>
<field idx="1" label="Phone" field-type="folder"
required="false" type="String"/>
<field idx="2" label="Contact" field-type="folder"
required="false" type="String"/>
<entity-acl>
```

```
<read-list>
<group>admins</group>
<group>publishers</group>
<group>editors</group>
<group>readonly</group>
</read-list>
<write-list>
<group>admins</group>
</write-list>
<write-doc-list>
<group>admins</group>
</write-doc-list>
<create-list>
<group>admins</group>
</create-list>
<del-list>
<group>publishers</group>
</del-list>
<security-list>
<group>admins</group>
</security-list>
</entity-acl>
<entity-type name="Product" title-index="2"
doc-title-index="9" inherit-parent="true">
<index idx="0" label="Title" index-type="folder"
required="true" type="String"/>
...
<entity-type name="Deal"
title-index="3" doc-title-index="9"
inherit-parent="true">
<index idx="0" label="Title" index-type="folder"
required="true" type="String" />
...
```

```
</entity-type>
</entity-acl>
</entity-type>
</dms-entity-model>
```

Appendix D: DocManager Wrapper API

This section exposes the benefits of the DocManager wrapper API as opposed to the DocManager EJB API.

This section contains the following topics:

DocManager Wrapper API Overview

In addition to the direct use of the DocManager EJB API (such as, through the use of Resource EJB), an easier to use wrapper API has been introduced in order to simplify interactions with DocManager.

Other benefits inherent in the new approach include standardized error handling, standardized validation checks, and no direct client EJB contact.

> **NOTE:** Usage of this wrapper API is strongly recommended since the DocManager EJB API has been deprecated for client use. Its use is also mandatory when linking DocManager to an external document management system.

Making Calls Simpler

Direct calls to DocManager EJBs sometimes require a large amount of code. The wrapper API encapsulates all calls to DocManager EJBs in an easy-to-use API so that client interactions with DocManager consist of as few steps as possible for each task.

Error Handling

The wrapper DocManager API wraps all EJB exceptions and throws DocManager specific exceptions. These exceptions provide a status code, a user-friendly message, and also contain the root cause of an error.

Permissions exceptions can be handled independently.

Validation

A number of the methods in the wrapper interface perform additional validation checks, such as verifying that a resource does not already exist, or that the operation attempted has not been disabled. In each case an appropriate error is thrown.

Interactions with the External DMS

All necessary interactions with DocManager and the external DMS are internally handled by the wrapper API when Scrittura is integrated to an external DMS or uses DocManager in its file system configuration. No extra method call either to DocManager or the external DMS is therefore necessary.

Using the Wrapper API

This section details how to use the wrapper DocManager API.

Interface Location

All classes related to the wrapper API interface are located in the `com.ipicorp.docmgr.docmgrinterface package.`

`ResourceData, FileData,` and DocManager exception classes are located in `com.ipicorp.docmg.util`.

Calling the Wrapper API

`DocmgrResourceInterface` is the parent interface of the wrapper API. Resource-type-specific functionalities (for folders, documents, and links) are accessible through the following interfaces, which extend `DocmgrResourceInterface`:

- `DocmgrFolderInterface`
- `DocmgrDocumentInterface`
- `DocmgrLinkInterface`

Implementations of those interfaces are accessed using the `DocmgrInterfaceFactory` class.

Example

```
DocmgrInterfaceFactory docmgrInterfaceFactory
= new DocmgrInterfaceFactory();
DocmgrDocumentInterface docmgrDocumentInterface
= docmgrInterfaceFactory.getLocalDocumentInterface();
```

Code Samples Using the Wrapper API

This section contains code samples illustrating common use cases. A basic description of the interfaces is provided as well as examples of the processes for creating resources and adding versions. For other operations, please refer to the Javadoc provided with the Scrittura release.

> **NOTE:** Unless specified otherwise, code extracts quoted here do not contain exception and transaction handling.

Document Creation and Version Creation

The following examples provide code extracts for document and version creation, using both the EJB API and the wrapper API. As demonstrated by those examples, document and version creation have been drastically simplified with the wrapper DocManager API.

Example using the deprecated EJB API

```
ResourceLocalHome home

= IpiDocmgrFactoryUtil.getFactory()

.getResourceLocalHome();

Collection col

= resourceLocalHome.findByParentIdResourceTypeAndIndex0

(folderId, ResourceTypes.RT_DOCUMENT, title);

if (col.size() == 0)

{

document = resourceLocalHome.createDocument(folderId,

title);

ResourceData data = document.getData(); data = setIndices(data, etype, title); data
= setFields(etype, data);


}

else

{


document.setData(data);


document = (ResourceLocal) col.iterator().next(); document.addNewVersion();

}

document.lockCurrentVersion();

FileData fdata = new FileData(); fdata.data = fileContent; fdata.extension =
extension;

document.putFile(fdata, modifiedBy);

ResourceHelperSession rh = IpiDocmgrFactoryUtil.getFactory()

.getResourceHelperSessionHome().create();

if (rh.isFullTextSearchEnabled())
```

```
{
document.checkAndPutFullText(fileContent);
}
ExportUtilities expUtil = ExportUtilities.getInstance(); Integer DocResId =
document.getId();
if (expUtil.isUsed())
{

if (!expUtil.isMigratedToFileSystem(document))
{
String fsPath = expUtil.getFileSystemPath(document);

}
else
{
}
}

expUtil.migrateAllVersionsToFileSystem(document); log.info("Res. " + DocResId + "
migrated to "
+ fsPath);
log.warn("Res. " + DocResId
+ " is already migrated");

document.unlockCurrentVersion();
```

Example using the wrapper DocManager API

```
DocmgrInterfaceFactory docmgrInterfaceFactory
= new DocmgrInterfaceFactory();
DocmgrDocumentInterface docmgrDocumentInterface
= docmgrInterfaceFactory.getLocalDocumentInterface();
FileData fileData = new FileData(); fileData.setData(fileContent);
fileData.setExtension(extension);
// NOTE: the next call is optional and only for custom metadata; fileData.setIndices
(indices);
```

```
fileData.setFields(fields); fileData.setFullTextSearchEnabled(true);

ResourceData data = documentInterface.get(parentId, fileName); int id = (data ==
null)

? documentInterface.create(parentId, fileName, fileData)

: documentInterface.addVersion(data.getId(),

fileData).getId();
```

Folder Creation

The following examples provide code extracts for folder creation using both the deprecated EJB API and the wrapper API.

Example 1 using the deprecated EJB API

```
ResourceLocalHome home

= IpiDocmgrFactoryUtil.getFactory()

.getResourceLocalHome();

EntityType et

= IpiDocmgrFactoryUtil.getFactory().getSystemConfigHome()

.create().getEntityType(entityTypeName);

parent = home.createRootFolder(entityTypeName, folder); ResourceData rdata =
parent.getData(); rdata.index[Integer.parseInt(et.titleIndex)] = folder;
parent.setData(rdata);
```

Example 1 using the wrapper DocManager API

```
docmgrFolderInterface.create (DocmgrConstants.NO_PARENT_ID,

folderName, entityTypeName, null);

Example 2 using the deprecated EJB API

folder = resHome.createChildFolder(etype, parent.getId(),

title);

ResourceData data = folder.getData(); data = setIndices(…);

data = setFields(…); folder.setData(data);
```

Example 2 using the wrapper DocManager API

```
final ResourceData data = new ResourceData(); data.setIndices(indices);
data.setFields(fields);

docmgrFolderInterface.create(parentId, folderName,

entityTypeName, data);
```

Setting Indexes and Fields Using the DocmgrConfigInterface

The `EntityType` class in `com.ipicorp.docmgr.util` should not be used directly with the wrapper interface. The DocmgrConfigInterface provides a number of methods to retrieve the appropriate information. Since DocManager handles the title index internally on document creation, this should rarely be used.

```
Example using the deprecated EJB API

EntityType et

= IpiDocmgrFactoryUtil.getFactory()

.getSystemConfigHome().create()

.getEntityType(entityTypeName);

if (et.getTitleIndex().equalsIgnoreCase("fixed"))

{

resourceData.index[0] = parentName;

}

else

{

resourceData.index[et.getTitleIndex()] = parentName;

}


resourceData.index[0] = parentName;

resourceData.index[et.getTitleIndex()] = parentName;
```

Example using the wrapper DocManager API

```
int folderTitleIndex = docmgrConfigInterface

.getFolderTitleIndex(entityTypeName); resourceData.index[folderTitleIndex] =
parentName;
```

DocManager Constants

The wrapper API also contains a new interface, DocmgrConstants, which holds some hard-coded string information relevant to the client. These include the root folder name and id, the index where the document name is stored, and the error codes returned in DocManager errors.

Send documentation feedback

If you have comments about this document, you can contact the documentation team by email. If an email client is configured on this system, click the link above and an email window opens with the following information in the subject line:

Feedback on Micro Focus Scrittura 4.4.10.5 Administration Guide

Add your feedback to the email and click **Send**.

If no email client is available, copy the information above to a new message in a web mail client, and send your feedback to autonomytpfeedback@microfocus.com.

We appreciate your feedback!