Tahiti

Administration Guide

Tahiti: Administration Guide

5.x

Table of Contents

Chapter 1. Introduction

Application Tahiti provides access to the documents stored in the document management system. Tahiti allows to view and edit such documents as well as to create new ones. Program is also design to be entry point to the electronic archive. Tahiti have support for:

- document scanning (flat scanners, mid- and high-speed scanners, quality control, bar-codes, etc.)
- digital cameras
- document identification
- emails

Whole application consists from main program and libraries which provides integration with other applications like Microsoft Office, Internet Explorer, Lotus Notes.

This guide describes technical features, application configuration and customization. It is intended for administrators, technical support and other specialist. This is not user guide for Tahiti. It is also possible to integrate Tahiti with other third-party applications or use from other application. For these options look at the programmers guide or contact software vendor.

This documentation evolves in the time. Please let us know if you find any mistake or inconsistency in this documents. Email: <tahiti@lightcomp.com>

Chapter 2. Installation

1. Software and hardware requirements

Program is designed for Microsoft Windows 2000, XP, 2003, Vista, 2008 and later editions. Tahiti is fully 32-bit application. Program can be used together with terminal services on Windows 2000, 2003, 2008, Citrix – there are no explicit restrictions but some can exists which are given by these systems (e.g. color depth). Usage of scanners and digital cameras together with terminal servers is not recommended and can be problematic.

1.1. Hardware

Hardware requirements depends on used operating system. It also depends on the way of Tahiti usage (workplace type), document types and their sizes. For scanning workplace's consult hardware requirements with scanner manual and also take a look in the chapter about scanning.

Table 2.1. Minimal configuration

Processor	Pentium II 400+
RAM	256Mb, recommended 512Mb
Graphical card	1024x768x16bits

Appropriate pointing device is essential. Program is design to run in dual head configuration (two monitors). It is usual to run application for access to the information system on one monitor and Tahiti on the second one. Application support such usage.

Tahiti can be resource consuming application. It depends on way of usage but it can be caused by:

- parallel opening of higher number of pictures (photos)
- work with embedded objects (OLE2)
- scanning with some transformations, smart quality control, bar-code recognition etc.

Note

Tahiti is graphical application and it is recommended to have enough system resources for efficient usage. It is also very important to train users for proper use of Tahiti. We know from our technical support what users are able to do. So please do not underestimate good training.

1.2. Microsoft Windows

Tahiti is design to be run on Microsoft Windows 2000 and later. From version 5.x Windows 2000 are not fully supported and we recommend to use Windows XP or later. Tahiti is registered in the system as a COM Server and requires proper registration. Check installation section for more information about required libraries and their registration.

1.3. Linux + Wine

Tahiti can be used on the Linux platform together with Wine. However some features are only supported on Microsoft Windows. Please contact software vendor for proper configuration and further informations.

2. Distribution

Tahiti is distributed in two forms:

- MSI package (Microsoft Installer) - end user installation file
- ZIP file - contain all Tahiti files and also SDK

You can create your own customized distribution/installation package - based on MSI or use othe installation program (InstallShield, InnoSetup). Such package have to comply with License Agreement.

2.1. MSI Package

Tahiti starting from the version 5.0.4 is distributed as MSI package. It is currently preffered way of distribution. One package can be used for end-user installation and also can be customized by administrator. Check chapter MSI Customization for more details about MSI customization.

2.2. ZIP file

ZIP file contains binary files, documentation, configuration files, examples, add-ons and also SDK (Software Development Kit). File can be used by developers (to use SDK) or by system integrators and administrators to create own custom install. Following table shows distribution layout.

Table 2.2. Distribution layout

Path	Description
`/bin`	binary files, main application
`/bin/chrome`	Chrome files for Tahiti and Xulrunner
`/bin/components`	xpcom components
`/bin/configs`	configuration directory
`/bin/defaults`	Xulrunner default configuration
`/bin/greprefs`	Tahiti-Mozilla preferences
`/bin/locale/??/LC_MESSAGES`	localization packages, cs - Czech, ru - Russian
`/bin/logs`	log files are stored in this directory (if enabled)
`/bin/scannerui`	scanner control dialog
`/conf`	example of configuration
`/doc`	Documentation (.chm, .pdf)
`/doc/admin`	Administration guide
`/doc/pg`	Programmers guide
`/doc/scan`	Separators for scanning
`/doc/sap`	examples of SAP integration
`/ext/..`	Tahiti extensions
`/ext/barcode`	Bar code recognition, need separate license
`/ext/fastReport`	FastReport
`/redist`	Redistributation files (runtimes required by Tahiti)
`/sdk`	Software Development Kit for Tahiti
`/sdk/addons`	Addons for 3rd party applications (Lotus Notes)
`/sdk/bin`	Binary files for SDK (tests, utilities)
`/sdk/samples`	Examples of various types, scripts, etc.
`/sdk/samples/DataForms`	Examples of data forms - used for data-mining

Path	Description
/sdk/samples/forms	Examples of Tahiti simple forms
/sdk/samples/packages	Examples of Tahiti Packages (.tpkg)
/sdk/samples/sap	Examples of SAP integration objects
/sdk/samples/vbscript	Examples of Visual Basic Scripts (.vbs)
/sdk/xpidl	XPCOM interface definition files (.idl)
/setup	Default setup script for InnoSetup
changes.html	list of changes

To prepare your own installation package you should consider using of MSI based installer. You can use classical installation products like InnoSetup. In such case it is up to you to prepare distribution plan and distribution itself. However there are pre-prepared setup configuration files. For larger organizations we recommend to use Invoker for distribution and installation. Invoker is separate application for automatic updates and installations.

3. Custom Installation

You can skip this section if you are using MSI based installer.

Installation depends on way of usage and customization for given workplace. In general you should install all files from the bin directory and include also other files like your own extensions and add-ons. Description of all files is in separate paragraph. Default and recommended directory for installation is Program Files\LightComp Tahiti 5\ . User configuration should be done in *domain configuration* files, these are installed in the configs directory.

Some of the files have to be registered during installation.

Table 2.3. Files to be registered

Name	Description
/bin/BarcodeControl.dll	Component for barcode generation
/bin/ExcelAddin.dll	Microsoft Excel integration
/bin/IEHelper.dll	Internet Explorer integration
/bin/OutlookAddin.dll	Microsoft Outlook integration
/bin/TahitiPackage.dll	Component used for document processing and automation
/bin/TahitiRedemption.dll	Component for MSG conversion
/bin/WordAddin.dll	Microsoft Word integration
/bin/Components/TahitiSapLink.dll	Component for SAP integration
/bin/Tahiti.exe	Main application (run **Tahiti.exe /Register**)

4. Running Tahiti

Tahiti can be run from main directory clicking on the file Tahiti.exe or using COM interface (detail description is in the Programmers Guide). User have to select domain, offline or on-line mode. It is not possible to send or receive any data in offline mode. Tahiti is usually executed without any parameters.

Tahiti will alwasy run as 1 process for several domains.

4.1. Parameters

Table 2.4. Tahiti parameters

Parameter	Description
/d <domain>	Domain to be used.
/offline	Automatically run Tahiti in offline mode.
/password <password>	Password
/profile <name>	Use specified profile. This option can be used to force Tahiti to use profile. 'name' is name of file profile as defined in the `profiles.xml`. E.g.: `/profile tahitiDefault` will read configuration from file tahitiDefault.xml. Profile will be read only if domain is specified.
/user <user-name>	User name

4.2. Registration

During the installation process Tahiti have to be properly registered. All required .dll files have to be registered and also main binary. To do this step there is set of parameters. Do not use this parameters together with regular parameters.

Table 2.5. Tahiti parameters

Parameter	Description
/Register	Register Tahiti.exe
/Unregister	Unregister Tahiti.exe
/Embedding	Run Tahiti in embedding mode (used for COM)
/Automation	Run Tahiti in automation mode (used for COM)

5. Local Repository

Tahiti requires local document repository where are stored documents, working files, user configurations. Repository is usually at the `Documents and Settings\USER\Application Data\LightComp\Tahiti\5.0\`. Some working files, cache etc are stored in the location of non-roaming profile, typically at `Documents and Settings\USER\Local Settings\Application Data\LightComp\Tahiti\5.0\`. Default location can be override in configuration, parameter `Tahiti.Repository.Path`.

5.1. Repository structure

Repository contains several files and directories. Files and directories bellow are part of the roaming profile. If user log on another computer within Domain (Active Directory) these files will be shared.

Table 2.6. Repository structure

Name	Type	Purpose
`settings.xml`	file	User settings
`<domain>/*.xml`	file	Domain specific configuration files, document types etc.
`<domain>/profile.xml`	file	Actual toolar/user interface layout

Name	Type	Purpose
cmserver2.xml	file	active copy of file defining document types - older format
cmserver3.xml	file	active copy of file defining document types - newer format
domain.xml	file	Configuration file for the domain. This is optional file - domain configuration can be read from this file or from file in the configs directory.
presentations.xml	file	active copy of file for displaying document hierarchy
DS	directory	scanned batches
EL	directory	history records
profiles	directory	tool-bar layouts
thumbnails	directory	thumbnails cache, part of the local (non migrating) profile

Directory can contain other configuration files downloaded from the server. These files have higher precedence then files in the configs directory.

5.2. Local part

Repository contains several files and directories. Files and directories bellow are part of the roaming profile. If user log on another computer within Domain (Active Directory) these files will be shared.

Table 2.7. Repository structure

Name	Type	Purpose
camel-cert.db	file	Certificates used by mail component
WFS	directory	Working files (temporary).
mozilla-sec	directory	Mozilla security database (keys, certificates).
<domain>/db	file	object database of local repository
<domain>/data	directory	contain binary data for documents in local repository, part of the local (non migrating) profile
<domain>/damis	directory	contain data for Damis system (downloaded packages).

6. File Formats

Tahiti have direct support for several document and image types. These are opened inside Tahiti and can be viewed and edited. Document type is determined by MIME type provided for given file from information system or by operating system. File types which does not have direct support are opened in external application.

It is essential to establish rules which defines supported MIME types and application used for such files. Tahiti can be configured in several ways. There is brief overview of supported types.

6.1. Images, photos

Tahiti can view and annotate several image types. Usually TIFF files are used for storing scanned images and JPEG files for photos. TIFF often serve as a container for several other file formats like CCITT G3, G4, etc. Tahiti have extensive support for TIFF variants.

Table 2.8. Image view supported types

Extension	MIME type	Description
bmp	image/bmp	Windows or OS/2 Bitmap
cut	image/freeimage-cut	Dr. Halo
g3	image/fax-g3, image/g3fax	Raw G3 fax
gif	image/gif	GIF image
ico	image/x-icon	Windows Icon
iff,lbm	image/freeimage-iff	IFF Interleaved Bitmap
jpg,jif,jpeg,jpe	image/jpeg	JPEG - JFIF Compliant
jp2,jpc,j2k	image/jp2	JPEG-2000 JP2 File Format and JPC Code Stream Syntax (ISO/IEC 15444-1)
koa	image/freeimage-koala	C64 Koala Graphics
mng	video/x-mng	Multiple Network Graphics
pcd	image/x-photo-cd	Kodak PhotoCD
pcx	image/x-pcx	Zsoft Paintbrush
png	image/png	Portable Network Graphics
pbm,pgm,ppm	image/freeimage-pnm	Portable Network Media
psd	image/freeimage-psd	Adobe Photoshop
ras	image/x-cmu-raster	Sun Raster Image
tga,targa	image/freeimage-tga	Truevision Targa
tif,tiff	image/tiff	Tagged Image File Format
wap,wbmp,wbm	image/vnd.wap.wbmp	Wireless Bitmap
xbm	image/x-xbitmap	X11 Bitmap Format
xpm	image/xpm	X11 Pixmap Format

6.2. Word, Excel and OLE2 servers

Tahiti is OLE2 container. It means it is possible to work with Word, Excel and other documents inside Tahiti. There is special support for Microsoft Office documents. Lot of OLE2 servers have small differences in behavior and might need some integration work. Fully tested and recommended are Microsoft Office versions 2000, XP, 2003, 2007, 2010.

6.3. Text Files

Text files are expected to be in local machine encoding. Information system should not use text files directly but preferable other file formats, like for example HTML for read-only documents.

6.. HTML, PDF files

Internet Explorer is one of embedded containers inside Tahiti. It allows to display HTML files but also PDF files and other read-only formats. Adobe Acrobat should be installed. It is possible to use other alternative applications which have plug-ins for Internet Explorer.

6.5. Forms

It is possible to display .TFR files (Tahiti Form). Tahiti has support for simple form processing. They can be used for data mining or other tasks where user input is required. There is special chapter about form processing and creation.

6.6. Packages, reports

Files with extensions TPKG. Such files are packed files which contains transformation script, document/ report template and instruction how to process. Such files are processed on the client side and can produced Word, Excel and other documents. Packages are often used for personalized letter and reports. There is special chapter on this topic.

6.7. Email's

Tahiti can be used to view and edit email's. It it possible to integrate writing/reading email's with work-flow events. Email's can be archived and identified same way as images and other documents. Email support is described in separate chapter. Tahiti in general handle mimetypes: `message/rfc822`, `application/msoutlook`. New-ly created or replied emails have mime-type `Tahiti.ComposeEmail`.

6.8. OpenOffice

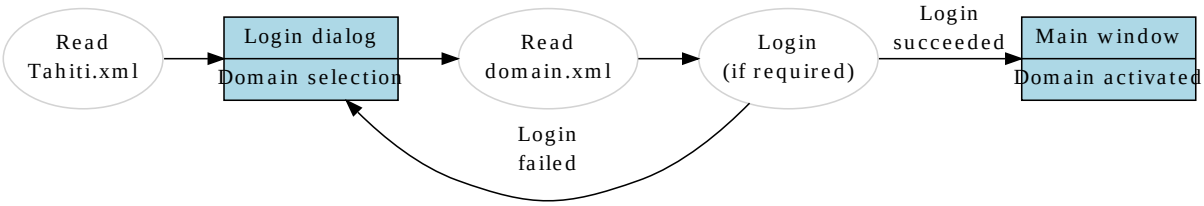
Tahiti can open OpenOffice file formats like ODT, ODS, ODP. Tahiti is tested with OpenOffice 3.1 and higher. All documents are opened inside Tahiti.

Chapter 3. Configuration

Tahiti cannot be run as a stand alone system but only in connection with some information system, integrated into business process or as part of document management system. Such system defines document types, document identifiers and relations. Documents are valid only in context of such system.

Tahiti can provide access to one or more information systems (so called domains) and every domain have to be described in separate configuration file.

Configuration also allows to customize Tahiti for several different workplace's and different tasks. Proper configuration and distribution of configuration files is also vital for effective usage. Configuration process is on the next picture.



Basic configuration is stored in the file `Tahiti.xml` which is located in the main directory. This is key file for Tahiti. Please do not remove any configuration options from this file. It is only possible to change default values. Any configuration changes should be done primary in the domain configuration files. Domain configuration files are stored in the directory `configs` or directly at the local repository in the file called `<domain>/domain.xml`.

```
\TahitiDirectory
  Tahiti.exe
  Tahiti.xml
  \configs
    DOMAIN1.xml
    DOMAIN2.xml
    \DOMAIN1
      ... DOMAIN1 configuration files ...
    \DOMAIN2
      ... DOMAIN2 configuration files ...
```

Every domain is represented by one configuration file in `configs` directory and one directory with subsequent configuration files for this domain. Name of the directory have to be same as domain name in configuration file.

Common configuration parameters for all domains:

- components - available components
- MIME types - description of document types, how to open, edit them

Every domain have to have these parameters:

- domain name - unique string for every domain. Usually in format of domain. e.g. primary.accounting.skoda-auta.cz
- communication - configuration of communication channels and their settings

All configurations are stored in xml files. Configuration is stored in pairs of "name" and "value". Value can be string or number.

1. Domain

Domain configuration file is usually named DOMAIN.xml (where DOMAIN is domain name) and is stored in `configs` directory. File can override almost all configurations specified in `Tahiti.xml`. Minimal domain configuration have to contain:

- domain name
- user domain name - domain name displayed to user

Each domain have separate working directory, local document database. All these items are stored in a directory call local domain repository. Domain configuration can be also stored directly in this directory and file have to be called `domain.xml`.

Example of domain configuration:

```
<?xml version="1.0"?>
<Config>
 <Item name="Connection.Domain"
       value="lc.ucto.lightcomp.cz"/>
 <Item name="Connection.Name"
       value="LightComp.Ucto"/>
 <Item name="Connection.Options"
       value="...."/>
 <Item name="Connection.RequestConfig.DocumentTypes"
       value="cmserver2.xml"/>
</Config>
```

Table 3.1. Domain configuration options

Name	Description
`Connection.Domain`	Domain name, used as a domain identification. It can be server domain name with added service name, e.g. documents.server1.skoda-auto.cz
`Connection.Name`	Domain name displayed to user, should be easy for user to understand where he is going to connect

If any of these two options is not defined configuration is invalid and cannot be used. List of valid configurations is displayed in the login dialog at Tahiti start-up.

1.1. XML format

`Tahiti.xml` has relativly simple format. Domain configuration has almost the some with only few small differences. Each configuration contains root tag <`Config`>. This tag contains list of items each within tag <`Item`>. This tag describe one option. Definition of such option have to be in the main configuration. This definition contains attributes `name`, `type` and `default`. Domain configuration on the other hand has only attributes `name` and `value`. Item definition in the main configuration can contain tag <`description`> with the description of the item.

Attributes of the tag <`Item`> in the configuration file are in the following table.

Table 3.2. Attributes of tag <Item>

Name	Usage	Description
`name`	`tahiti.xml`, `domain.xml`	Name of the item
`type`	`tahiti.xml`	Type of the item, can be `int` or `string`

Name	Usage	Description
default	tahiti.xml	Default value. This value is used for all domains if not redefined in domain.xml.
value	domain.xml	Domain specific value of the item.

2. Repository

Repository is created inside the user profile. It is possible to override this location in option Tahiti.Repository.Path. There are only few reasons for changing this location and we do not recommend to change it. Default location allow to use Roaming Profiles in Windows.

Table 3.3. Repository parameters

Name	Description
Tahiti.Repository.Path	Path to the repository. This have to be fully qualified path. Path can contain variables.

3. Configuration inside Repository

Domain configuration can be stored inside repository (not part of the configs directory) and all other configuration files can be downloaded from the server. This approach is suitable for the situation when there is one Tahiti distribution common for several domains. Domain configuration is stored at the file called domain.xml. It is possible to use utility called configDownload to download this configuration from the web server and store it at the right place.

There is special configuration option called Connection.RequestConfig.TahitiPacked. This is name of server side file with packed configuration. If this option is not empty configuration will be downloaded during logon. Tahiti is using MD5 sum to check if there is new configuration available.

4. Profiles

Profiles are fundamental part of Tahiti and allow to customize look&feel to the specific needs, e.g. document viewing, scanning. There are several predefined profiles and administrator can change these or create new profiles. Default profiles are located in the directory bin/TahitiProfiles. Main file is profiles.xml and contains description of defined profiles. All profiles can be also redefined in domain specific settings.

4.1. Default Profiles

There are 5 standard predefined profiles in Tahiti distribution.

Table 3.4. Default Profiles

Name	Description
tahitiDefault	This profile is used when Tahiti is run for first time. It can be later used to change or restore profile.
tahitiDocViewer	Profile for document viewing and editing.
tahitiScan	Profile is design for scanning documents using attached scanner.
tahitiPhoto	Import photos from the camera connected over USB. Only cammeras which are mounted as disk are directly supported, typicaly Olympus.
tahitiDamis	Profile for mass-data processing using Damis System. This profile is hidden by default.

4.2. User-Defined Profiles

It is possible to redefine any of the default profile or create new profile. Recommended way is to customize predefined profiles and only if no one is suitable to create new. User-defined profiles are defined in the `profiles.xml`. It is possible to change `profiles.xml` in the `bin/TahitiProfiles` but it is more convenience to change redefine this file in the domain specific directory in the repository, e.g. `<repository>/<domain>/TahitiProfiles/profiles.xml`.

There are two profile specific configuration options in the `Tahiti.xml`:

Tahiti.Profiles.Default	Name of the default profile. It is used for first time when user run Tahiti
Tahiti.Profiles.Active	Name of the last used profile.

4.3. Profile Reference

Profile Reference sections contains detail description of `profiles.xml`.

Example 3.1. Example of profiles.xml

```
<?xml version="1.0"?>
<TahitiProfiles>
 <Profile id="tahitiDefault"
    contentUrl="chrome://tahiti/content/intro.xul"
    saveChanges="0">
  <Name>Introduction</Name>
  <Name xml:lang="cs">Uvítání</Name>
  <Name xml:lang="en">Introduction</Name>
  <Description>This profile is used ....</Description>
  <Description xml:lang="cs">Profil použitý p#i ...</Description>
 </Profile>
 <Profile id="tahitiDocViewer">
  <Name>Documents</Name>
  <Name xml:lang="cs">Dokumenty</Name>
  <Name xml:lang="en">Documents</Name>
  <Description>Recommended profile for ....</Description>
  <Description xml:lang="cs">Zobrazení ....</Description>
 </Profile>
</TahitiProfiles>
```

Root Element: TahitiProfiles

4.3.1. TahitiProfiles

This is top-level container element for all defined/redefined profiles.

Children: **Profile**

4.3.2. Profile

The Profile element contains profile definition Each profile have to have attribute `id` - this is unique profile identifiier. Redefined profile have same `id` as its predecesor.

Children: **Name**, **Description**

Table 3.5. Profile Attributes

Attribute	Required	Description
id	yes	Profile identifier; this is the primary key for identifying profile.

Attribute	Required	Description
contentUrl	no	Optional attribute; this is URL of the file which will be opened when user select this profile. Default profile uses this option to open introduction page.
saveChanges	no	Option allows to control if user changes should be saved. Default value is 1; changes are automaticly saved. Possible values: 0|1
visible	no	Flag if profile is visible for users. Default value is 1; profile is visible. If this value is 0 profile is hidden. Possible values: 0|1

4.3.3. Name

The Name Element is profile name. This is user visible name and can be localized.

Inner Text: Name of the profile.

Table 3.6. Name Attributes

Attribute	Required	Description
xml:lang	no	Language; this is language for which is this name valid. If not specified inner text is used as default name.

4.3.4. Description

The Description Element is profile description. This is description of the profile.

Inner Text: Descritption of the profile.

Table 3.7. Description Attributes

Attribute	Required	Description
xml:lang	no	Language; this is language for which is this description valid. If not specified inner text is used as default description.

5. Logging

It is possible to log several application activities. It can be used for diagnostic and debugging purposes. It is recommend to switch off logging in production environment.

Table 3.8. Logging options

Name	Description
Tahiti.Log.Path	Path where to store log file. Path should include last slash otherwise it is used as a filename prefix.
Tahiti.Log.System	Logging level. =0 - switched off, >0 - switched on

Logging level should in interval <0; 4>. 4 is the highest logging level. Logging can be only set in the main `Tahiti.xml`.

6. Components

Tahiti consist from several components. Each component have own configuration options and can be switched on/off. There are internal components (shared components) which are integral part of the application (through some of them can be switched on/off) and also there are domain components. Domain components can be used as plug-ins to extend Tahiti.

Not all work-places need all components and they does not have to be initialized. Switching-off can speed-up application start-up and use less system resources.

Table 3.9. List of shared Tahiti components

Name	Description
`Component.Mozilla.Core`	Basic component for Mozilla (XPCOM, XUL) integration.
`Component.ExternalView`	Bridge for other XPCOM-based views
`Component.XULView`	View for XUL files
`Component.XULFormView`	View for files based on XUL (predecesor for mail view)
`Component.ComposeMail`	Mail composer
`Component.Mozilla.EmailView`	Base component for mail viewer
`Component.ViewMail`	Mail view component
`DocumentIO`	Document Input/Output Component
`Component.PictureView.Base`	Picture view base component
`Component.EditView`	Edit view component
`Component.FormView`	Form view component
`Component.HTMLView`	HTML view component (used to display PDF)
`Component.Ole2View`	OLE2 view component (.DOC, .XML, .PPT)
`Component.ExecutableView`	Component for openning outside Tahiti
`Component.PackageView`	Package view component
`Component.PictureView`	Picture view component (.TIFF, .JPEG, ,.PNG)
`Component.XPCOMSharedBridge`	Bridge for other XPCOM-based shared components

Table 3.10. List of Tahiti domain components

Name	Description
`Core.Repository`	Document repository. This component is essential for DocumentManager
`Core.Scan`	Low-level component for scanning. TWAIN interface
`PropertyControl`	Property Control - display list of properties
`IntegratedBrowser`	Optional bar with integrated browser (Firefox). Can be used for closer integration with other system
`DocumentManager`	Component provides access to localy stored documents
`Scanner`	Scanner component allows to scan document
`PhotoImport`	Component for import from cameras
`Damis`	Optional component, allow to communicate with DAMIS system
`Component.ShellFiles`	Optional component, allow to display embedded Explorer.
`CanClose.Component`	Component display confirmation dialog
`Remove Old Documents`	Component removes offline documents from the local repository during start
`License Component`	Check valid license and display dialog
`System Message Handler`	Component process action object 'Tahiti.System.Message'

Name	Description
Component.Tahiti.SapGui.ComBridge	External component which is used for SAP integration
XPCOMDomainBridge	Bridge for other XPCOM-based domain components

Tahiti can be extended with other components. There is configuration option called Components.XPCOM.Load. It is expected to receive comma separated list of components to be loaded. Components are physically stored inside directory Components. This option contains only domain components and cannot contain any shared components.

Example 3.2. Example of Components.Load

```
<Item name="Components.XPCOM.Load"
      value="@lightcomp.com/tahiti/sapgui.com;1">
</Item>
```

7. Communication

This chapter discuss communication between Tahiti (Document Manager) and other systems like document management system or information system. This chapter does not cover communication realized with other components and systems like DAMIS, Web, etc. These systems are described in separate chapters.

Tahiti allow communication primary with one document management system. It is called as Primary Connection. This connection should provide available document types, configuration. There is also possibility to communicate with more servers or use different communication protocol. Default and also the most powerful protocol is called DEP2. Protocol specification is available at the LightComp development web site. Each communication channel is configured by two items: used module name, configuration string.

Table 3.11. Communication configuration

Name	Description
Connection.Primary.Options	Settings of primary communication
Connection.PrintServer.Type	Type of module used for communication with printing service
Connection.PrintServer.Options	Setting for printing service

PrintServer connection can be used for sending documents to the specialized printing service. This logic should be preferably realized on the server side but if it is necessary it can be done on the client side.

Table 3.12. List of communication module

Name	Description
TAHITI_DEP2	Default communication protocol. This full-duplex protocol which allows server to push documents and events to the client and also to send changes and new documents from the client to the server
TAHITI_FTP_SEND	Standard FTP protocol. Allow to store new documents and changes to the server
TAHITI_SFTP_SEND	Standard SFTP protocol. Allow to store new documents and changes to the server

It is possible to write custom communication modules and customize Tahiti usage. Communication is configured with configuration string.

Example 3.3. Example of configuration string

```
user=%TAHITI_DOMAIN%:%TAHITI_USER%;
  password=%TAHITI_PASSWORD%;
  server=sds.lightcomp.cz;
  port=7780; pingInterval=10;
```

Configuration string consist from pairs (name=value) separated by semicolon. Configuration is specific for each module.

7.1. TAHITI_DEP2

Main communication module for connection to the Document Management System. This module allow to send and receive documents.

Table 3.13. DEP2 parameters

Parameter	Required	Description
user	yes	user name
password	yes	password
server	yes	server address - name or IP address
port	no	port where to connect, default port is 7780
logLevel	no	logging level, default is 0, 0 - switched off, 4 - maximum level
logTarget	no	where to log, it is possible to redirect all logging from file to the window`logTarget=window` will set write data to the separate window
cache1	no	Path to the local cache. Cache is created in the subdirectory "pageCache". If path does not exists it will be created.
cache1NumLevels	no	Cache is using hierarchy structure of directories. There is 16 directories on each level. Default value is 1.
cache1MaxSize	no	Maximal cache size. Size is in the Megabytes (MB). If this parameter is not specified cache size is unlimited.
cache2	no	Path to the global cache. This cache has 3 levels. If this path is not valid at the Tahiti startup-time cache will not be used.
ssl	no	SSL based communication, default is 0, 0 - switched off, 1 - switched on
pingInterval	no	Time in seconds between keep alive packets. By default keep alive packets are not sent.
channels	no	Set if channels should be used for sent data, default is 0. 0 - switched off, 1 - switched on
filter	no	Name of attribute to be used as a filter. If set only documents with this attribute will be sent.
connectOnSend	no	Usually connection is established when Tahiti starts up. This options allows to start/stop connection for each sent document (same as HTTP), default is 0. 0 - switched off, 1 - switched on
auth	no	Authentication method. Default value is plain. (This parametr can be overwriten when Tahiti is run via COM interface)

Cache is using auto-management technique. Maximum cache size is checked 10 minutes after start and later checked every hour.

Example 3.4. Example string

```
user=%TAHITI_USER%;password=%TAHITI_PASSWORD%;
  server=192.168.0.1;port=7780;
  logLevel=4;logTarget=window
```

Example 3.5. Example with local cache

```
user=%TAHITI_USER%;password=%TAHITI_PASSWORD%;
  server=192.168.0.1;port=7780;
  logLevel=4;
  cache1=%CSIDL_LOCAL_APPDATA%\LightComp\Tahiti\5.0\%TAHITI_DOMAIN%
```

8. Variables

Tahiti have system of internal variables. It is possible to use such variables in several configuration strings. Table contains some broadly available variables. Other variables are valid only in special cases.

Table 3.14. Variables

Name	Description
TAHITI_USER	user name - name of logged user
TAHITI_PASSWORD	password
TAHITI_REPOSITORY	path to the repository
TAHITI_DOMAIN	domain
TAHITI_BIN_PATH	path to the binaries
CSIDL_DESKTOPDIRECTORY	desktop directory
CSIDL_PERSONAL	documents folder
CSIDL_APPDATA	application folder
CSIDL_LOCAL_APPDATA	local application folder (not part of roaming profiles)
CSIDL_PROGRAM_FILES	program files

Note

Variables CSIDL_XXX have same meaning as corresponding variables as defined in the Windows Shell and Controls API™.

9. MIME types

Document processing in Tahiti is based on the MIME types. They are used for decision how to view and edit page. It is also possible in some cases find out corresponding MIME type from extension. Configuration files are used to set mapping between extension, MIME type and eventually to configure external application which is able to open it.

MIME type configuration is read in 3 stages:

1. freedesktop.org.xml

2. registry

3. mimetypes.xml

The best approach is to view and edit file inside Tahiti. It allows to have control over user actions and also to know which application is used and how. If file format is not supported for direct opening it is stored in temporary directory and opened with default system application associated this file type.

9.1. Freedesktop.org.xml

File contain platform independent list of MIME types and usual extensions and is used in the Tahiti as basic configuration. Actual file can be downloaded from http://freedesktop.org. There is also available specification of used file format. This .xml file is relatively big and it is possible to strip down unused types and speed up Tahiti startup.

Registry records are read after freedesktop.org.xml. Registry are also used for finding which application should be used for given MIME type, extension.

9.2. MimeTypes.xml

File is loaded after all other configuration files and can be used for overriding registry settings.

Use cases:

- specification of user defined types
- mapping between extension and MIME type
- configuration of preferred external application

Tahiti monitor each running external application and try to recognize when user close program and save modified file. MimeTypes.xml allow to configure technique of monitoring specific application.

Table 3.15. Monitor types

Type	Description
1	File lock is monitored
2	Process existence is monitored

File contain list of tags called `MimeType`. Application used for opening file can be defined inside this tag. If no application is specified tag is used only as mapping between MIME type and extension.

Table 3.16. MimeType Attributes

Attribute	Required	Description
type	yes	defined MIME type
extension	yes	extension for MIME type
attr	no	used monitoring technique

Example 3.6. MimeTypes.xml example

```
<?xml version="1.0"?>
<MimeTypeDatabase>
 <MimeType type="application/msword" extension=".doc"/>
 <MimeType type="text/plain" extension=".txt" attr="2">
   %SystemRoot%\system32\NOTEPAD.EXE %1
 </MimeType>
</MimeTypeDatabase>
```

It is possible to use variables in application specification. Variable `%1` will be substituted with file name.

10. Help Customization

Help files can be defined as part of configuration process. Help can consist from files distributed together with application as well as links and URLs. Content of menu Help is defined in the Tahiti.xml. There can be up to five user defined items.

Each item have to have name and associated command. Configuration values: `Help.User.X.Name` - name, `Help.User.X.Command` - command to be executed. X is number in the interval <1, 5>.

Example 3.7. Help Customization

```
<Item name="Help.User.1.Name" type="string" default="&amp;amp;Usage">
 <Description value=""/>
</Item>
<Item name="Help.User.1.Command" type="string"
   default="%TAHITI_BIN_PATH%\help\index.html">
 <Description value=""/>
</Item>
```

Example will create in the menu "Help" item "Usage". If user selects this item file `help/index.html` from the Tahiti directory will be open.

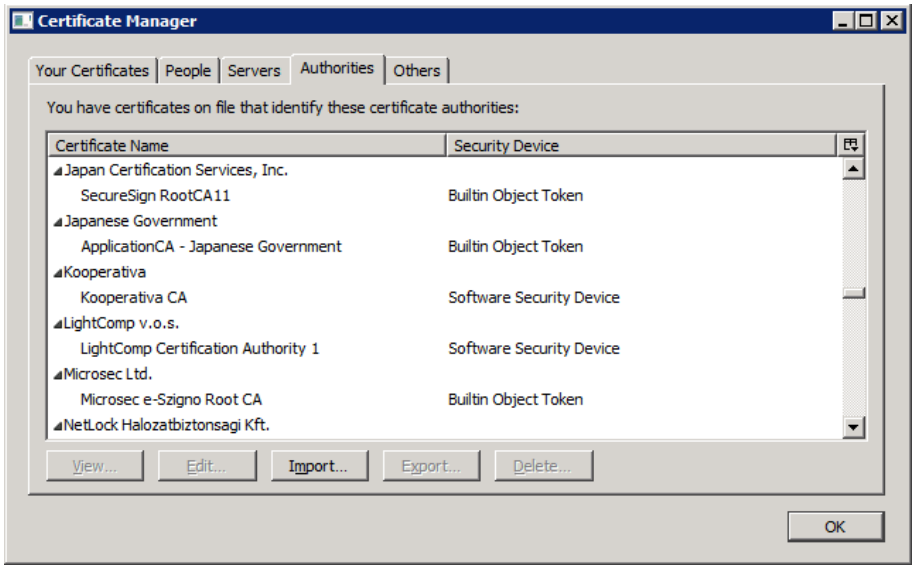
11. Root Certificates

Tahiti contains list of security certificates. There are several built-in root certificates (same certificates as included in the Firefox). It is possible to add to the Tahiti own root certificates. These certificates are defined in the file `TahitiCerts.xml`. File contains certificates in PEM format (BASE64 encoded certificate).

Example 3.8. TahitiCerts.xml

```
<?xml version="1.0"?>
<Certificates>
 <Certificate name="LightComp">
-----BEGIN CERTIFICATE-----
MIIDkTCCAnmgAwIBAgIICz4KhsYIKMcwDQYJKoZIhvcNAQEFBQAwVjEsMCoGA1UE
....
Y6cMS/Y=
-----END CERTIFICATE----
</Certificate>
</Certificates>
```

File can contain several certificates, each closed in tag `<Certificate>`. Attribute name is used to distinguish one certificate from other. List of installed certificates is available in the Certificate Manager. Here you can check and manage certificates.



Chapter 4. Documents

Tahiti works primary with structured documents. Document in Tahiti is defined by document type, unique identifier, collection of pages and attributes. Document attributes are defined as part of document type definition. Document management system have to provide list of available document types and their definitions. This list is stored on the client side inside repository in the file `cmserver2.xml` or `cmserver3.xml`. File is usually downloaded when user connect to the system. When application is offline last downloaded file is used.

Configuration files for document management system:

- `cmserver2.xml` - document types and attributes (legacy system)
- `cmserver3.xml` - document types definition (recommended version, used for new systems)
- `presentation.xml` - define views and document structure

All document types which can be displayed in the Tahiti have to be defined in the definition file. Undefined document types cannot be displayed nor processed.

There is example of document hierarchy and types in the Tahiti distribution. It is definition of documents in the accounting system.

1. Document Types

Each document stored in the document management system must have assigned document type.

Document type definition consist from several items:

- document type identification
- document name
- flag if new documents can be created
- class attributes - attributes associated with document type, these attributes are read-only
- document attributes - attributes associated with document instance

Document name should be clearly understandable for user. Prefix in document name is often used for easier document type selection - e.g. "I0 - Invoices Out". Deprecated document types can be marked with flag as non-creatable, such types are valid for viewing but not for creating new documents.

1.1. Class Attributes

Class attributes are attributes associated with document type which are read-only for document. This attributes can be used for document classification, description, sorting etc.. Example of class attribute: "Department" with values "Research", "Marketing", "Public Relations".

Class attribute is associated with document type and has defined value.

1.2. Document Attributes

Document attributes are created when document instance is created. New attribute is initialized with empty value and is identified by name. Attribute definitions are in the document type definition file

(`cmserver2.xml, cmserver3.xml`). It is also possible to specify rules for attribute values - like mask, attribute length. All attributes are transferred together with the document.

2. Format of cmserver2.xml

cmserver2.xml is obsolete format for exchanging document types. File format definition is part of the Programmers Guide.

3. Format of cmserver3.xml

cmserver3.xml is recommended format for exchanging document types. File format definition is part of the Programmers Guide.

4. Document Tree

Documents on the client side are organized in the tree. Structure of this tree is defined in the file `presentations.xml`. Tree should allow users to have a good orientation in the documents and quickly find required information. Tahiti support several document hierarchies and user can switch between these views on the fly.

Document hierarchy is based on the attributes and their values. Attributes are defined in the `cmserver2.xml`or `cmserver3.xml`.

4.1. Document Attributes

There are several system attributes which are defined for each document in the tree.

Table 4.1. Document Attributes

Attribute	Description
name	Document type name
Document.id	Document identifier, identifier is received from the server. New document has empty value.
Document.id_internal	Document identifier in the Tahiti. This value has runtime specific value. Value can be used only to distinguish between documents.
Document.version	Document version. Value is received from the server
Document.serverVersion	Document version. Second version value can be used for internal server operation. Client should not depend on this value
Document.readonly	Document read-only flag. • 0 - document has read/write access • 1 - document is read-only
Document.type	Identifier of document type. Document types are defined in the `cmserver3.xml`. This attribute have all time valid value.
Document.modified	Modification flag. • 0 - not modified • 1 - modified

Attribute	Description
Task.name	Name of connected task. This attribute can be used only in the presentation to check if document is part of task.

4.2. Filters

It is possible to use one or more trees to display documents. Number of used trees can be adjusted by setting variable `DocTree.TreeCount`. By default one tree will be used. Each tree can be used for different kind of documents or use same view.

5. File Format of presentations.xml

File contain document tree definition and definition of views. One document hierarchy is called "Presentation". File contain:

- Simple hierarchy - used for document type selection
- Dynamic views - used for document organization in the tree

5.1. Simple hierarchy

Simple hierarchy is used for document type selection and organization of document types. Organization is based on the class attributes and their values.

Example 4.1. Example of simple hierarchy

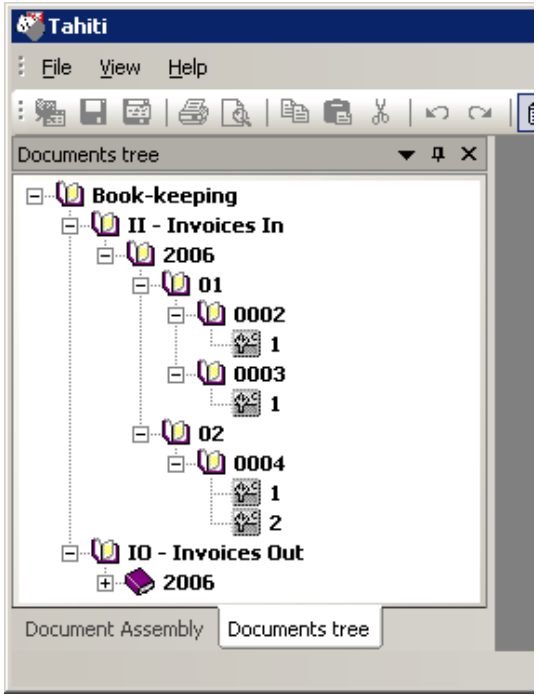
```
<?xml version="1.0"?>
<ViewSettings>
  <BasicHierarchy>
    <HierarchyItem name="BOOK_KEEPING__AREA"/>
  </BasicHierarchy>
</ViewSettings>
```

Simple hierarchy definition is stored inside the tag `<BasicHierarchy>`. Tag contain list of `<HierarchyItem>` with defined attribute `name`. This attribute have to contain class attribute value used for sorting. Such attribute should be defined for all document types.

5.2. Views

Views are defined in section `<Presentations>` in the `presentations.xml`. View can be defined one for all document types but also specialized for each one. View define levels in the tree and displayed values.

Example 4.2. Tree example



Example 4.3. Example of presentations.xml

```
<?xml version="1.0"?>
<ViewSettings>
 <BasicHierarchy>
  <HierarchyItem name="BOOK_KEEPING__AREA"/>
 </BasicHierarchy>

 <Presentations>
   <Presentation name="Area/Year/Period">
     <Document type="Tahiti.rootDocument">
       <Level text="%BOOK_KEEPING__AREA%" cond="-"/>
       <Level text="%name%" cond="-"/>
       <Level text="%BUSINESS_YEAR%" cond="-"/>
       <Level text="%PERIOD%" cond="%PERIOD%"/>
       <Level text="%FILE_NUMBER%" cond="-"/>
       <ShortDesc text="%Page.number%"/>
       <LongDesc text="%Page.number%-%CISLO_JEDNACI%"/>
       <VersionDesc text="Version-%Document.version%"/>
     </Document>
   </Presentation>

   <Presentation name="Year/Period/Area">
     <Document type="Tahiti.rootDocument">
       <Level text="%BUSINESS_YEAR%" cond="-"/>
       <Level text="%PERIOD%" cond="%PERIOD%"/>
       <Level text="%BOOK_KEEPING__AREA%-%name%" cond="-"/>
       <Level text="%FILE_NUMBER%" cond="-"/>
       <ShortDesc text="%Page.number%"/>
       <LongDesc text="%Page.number%-%CISLO_JEDNACI%"/>
       <VersionDesc text="Version-%Document.version%"/>
     </Document>
   </Presentation>

 </Presentations>
</ViewSettings>
```

Views are defined inside tag <Presentations>. Each view is identified by name and defined inside tag <Presentation>. Name is displayed in the user interface as the view identification. At least one tag <Presentation> have to be defined.

5.3. View

View definition can be one for all document types or specialized only for some types. Common view is defined inside tag <Document> with attribute `type="Tahiti.rootDocument"`. Specialization of view is done by setting attribute `type` to the document type.

Table 4.2. View definition

Tag name	Description
`Level`	Level definition, displayed in the tree. Level is displayed when associated condition is not empty.
`ShortDesc`	Page name, displayed in the tree
`LongDesc`	Page name, displayed as the view name
`VersionDesc`	Version name, displayed in the tree when more versions of one document are downloaded.

5.4. Level

Tag <Level> defines one level in the tree. Text displayed in the tree is defined in the attribute `text`. Condition if level will be displayed can be set in the attribute `cond`. Level is displayed if condition is not empty. Order of nodes in the same level can be set int the attribute `sort`.

Example 4.4. Level displayed in all conditions

```
<Level text="%NUMBER%" cond="-"/>
```

Example 4.5. Level with `sortBy` attributes, displayed only if %NUMBER% is not empty.

```
<Level text="Number - %NUMBER%" cond="%NUMBER%" sortBy="%NUMBER%"/>
```

It is also possible to define conditional level. Definition is on the next example.

Example 4.6. Conditional values

```
<Level>
  <Case text="%NUMBER%" cond="%NUMBER%"/>
  <Case text="%YEAR%-%PERIOD%" cond="%YEAR%%PERIOD%"/>
  <Case text="%YEAR%" cond="%YEAR%"/>
  <Case text="unknown"/>
</Level>
```

Tag <Case> contain same attributes as the <Level>, meaning is also same.

5.5. Filter

It is possible to possible only specific kind of documents in the view. Such approach can be useful to distinguish between newly created documents and received documents from the information system.

Document types:

- new document
- received document
- task document

Table 4.3. Filter types

Filter type	Description
0	Display all items
1	Display only new documents (created by user).
2	Display only existing documents (downloaded from the server).
3	Display only documents connected with the task. These documents have to be process by user and committed back to the server.
4	Display existing documents and tasks. Combination of filter 2 and 3.
5	Display new documents and tasks. Combination of filer 1 and 3.
6	Display new documents existing documents which are not part of any task. Combination of filer 1 and 2.

Used filter can be defined as part of presentation. There is optional parameter `filter`. If this parameter is not set no filter is used (all items are visible).

Example 4.7. Filter

```
<Presentation name="Area/Year - New documents"
                filter="1">
     ......
</Presentation>
```

Chapter 5. Workflow

Workflow is concerned with providing the information required to support each step of the business cycle. Tahiti can be used as client for such workflow system and do one step. For the current step, there may be multiple actions associated with document processing. Such document can be send to the Tahiti together with task definition and list of possible next steps.

Document can be sent to the Tahit in the read-only mode or with allowed modifications. Task can also define additional attributes which will be associated with document when it is finished and send back to the server in given state.

1. Native Format

Tasks description can be passed as standalone packet over communication channel or is usually packed in one of the document attributes - `Document.task.XML`.Next paragraph describes XML format of task and required attributes.

Example 5.1. Native Task Description (example)

```
<?xml version="1.0"?>
<DocumentTask id="task_15478" name="TaskName">
 <Attributes>
  <Attribute name="Description"
             value="Here is longer task explanation"/>
  <Attribute name="Priority" value="2007012816"/>
  <Attribute name="Creation" value="12-28-2006 14:30"/>
  <Attribute name="Expiration" value="01-28-2007"/>
 </Attributes>
 <Actions>
  <Action id="aid_1" name="Task Name"
     description="Choosing this state something might happen">
   <Attributes>
    <Attribute name="ActionResponse" value="Some value"/>
   </Attributes>
  </Action>
 <Actions>
</DocumentTask>
```

1.1. File Format

Root tag has name `DocumentTask`. There are two mandatory attributes `id`, `name`.

Table 5.1. <DocumentTask>

Attribute	Required	Description
id	yes	Task id, have to be unique identification of the task.
name	yes	Name of the Task. This value is directly displayed to the user and should be localized. Value is also used for task classification, it means that task of the same class should have same name.

Tag `Attributes` contains `Attribute` definitions. There is fixed list of possible attributes. Only these can be correctly interpreted by the Tahiti. `Attribute` is simple combination of `name` and `value`.

Tag `Actions` contains list of possible next states for the task. Each states have to be defined in the tag `Action`.

Table 5.2. <Action>

Attribute	Required	Description
id	yes	Action identificator. Used to distinguish between different actions.
name	yes	Name of the next state (action). This name is displayed to the user in the context menu. Value should be localized.
description	no	Action description, can be used to explain action and consequences of this state.

Each `Action` can contain list of attributes which are set when user select this action. When user select next state document is immediately send and user cannot change any attributes.

2. TPD Format

TPD is Tahiti Packed Document format. It is used as a container for one document or task from workflow system. There is one xml file, attributes and binary files. All data are packed inside one file - it is common zip file. There have to be at least one file called `Pruvodka.xml` which contain document, task and data description. All other files have to be listed inside main xml file.

Note

This file format is currently obsolete and should not be used for new systems.

Example 5.2. Pruvodka.xml (document without task)

```
<?xml version="1.0"?>
<PaperBack Area="POJ_UDAL" SentByUser="mposmurn"
       DateSent="20031104" Deletable="Y"
       Status="1" Draft="cmserver.xml">
  <Ids>
    <ID Name="CPS_CPU" Value="2030000010"/>
  </Ids>
  <Docs>
    <Document Name="009-Dopis" Deletable="Y" Archivable="Y">
      <Ids />
      <Pages>
        <Page Name="dopis.tpkg"/>
      </Pages>
    </Document>
  </Docs>
</PaperBack>
```

Root tag is named `PaperBack` and have to have these attributes:

- `Area` - Area for compatibility with Golem system, should be empty for other systems
- `SentByUser` - user name for document processing, can be empty
- `DataSent` - date when document was generated, only informative value
- `Deletable` - one of value Y|N, Y - document can be delete, N - document cannot be delete
- `Draft` - reference to the document system, currently unused

First format definition allowed to define more documents inside one file. It is no more possible and only one document can be stored inside TPD. Identificators can be set in two places. Upper level can contain three tags:

Table 5.3. TPD tags

Name	Description
Ids	Common identificators
Actions	List of available document actions. Used for task definition.
Docs	Documents definition. Currently can contain only one tag `Document`.

Document attributes:

- `Name` - document type name, this type have to be defined in the cmserver2.xml
- `Deletable` - currently unused tag, same meaning as `Deletable` on the main tag.
- `Archivable` - set if document can be stored in the archive, currently unused

Attributes are stored inside tag `Ids`, each combination of name, value in the tag `ID`. Attribute names are defined in the connected information system.

2.1. Actions

Section `Actions` contain list of target document states. After processing document and associated files user have to select one of the target states.

Example 5.3. Pruvodka.xml (Actions)

```
<Actions>
  <Action id="S1" name="Send Invoice"/>
  <Action id="S2" name="Cancel Invoice"/>
</Actions>
```

There are two defined actions in the example:

- Send Invoice
- Cancel Invoice

Each action have to have unique identifier and name. Name is displayed to the user. Identifier is used for processing conditional attributes.

2.2. Conditional Attributes

Task is defined as list of actions, target document states. Given state is signaled to the server by setting attribute value. Such attributes are called conditional attributes. Values of these attributes are set when user select new state. Consequently is documented submitted to the server.

Example 5.4. Pruvodka.xml (Conditional Attributes)

```
<Ids>
  <ID Name="NewDocState" Value="2030000010" OnAction="S1"/>
  <ID Name="NewDocState" Value="2030000021" OnAction="S2"/>
</Ids>
```

Conditional attribute is almost similar as regular attribute. Only one difference is attribute called `OnAction`. This attribute signals on which selected state should be value applied. Value is not used if another state is selected.

Chapter 6. Scanning

Tahiti has wide support for scanning and supports desktop, mid-range and also hi-speed scanners. It is important to carefully prepare scanning process. Such process have to be cost/time efective. There are lot of different scenarios and Tahiti can be easily accomodated for such process. Scanning is often directly connect not only with the document digitalization but also with document identification - attaching document type, filling attributes and sending document to the document store.

Some of the possible scanning scenarios:

- scan documents and store on the local disk
- scan documents, attach attributes and send to the document store
- scan documents, read attributes from the barcode and send to the document store
- scan documents and send them to the Chapter 7, *Document Assembly* system.

Used scenario depends on the type of the scanned documents, amount of documents and other conditions. It is also possible to use combination ot these methods.

Important part of the efective scanning process is to use predefined Section 1, “Scanning profiles”. These profiles are stored in stand-alone xml file which can be distributed to the users. Profiles can be also used to further automate document processing, set barcode recognition options and generation of attributes.

1. Scanning profiles

Scanning profiles are defined in the xml file. This file is part of the configuration and is called `scans.xml`. File has to be in one of the following locations:

- `<tahiti-dir>/locale/<locale>/scans.xml`
- `<tahiti-dir>/configs/<domain>/scans.xml`
- `<repository>/<domain>/scans.xml`

`scans.xml` can contain list of definitions common for all scanners and also individual list of definition for given scanner. Following example contains two profiles common for all scanners. First profile is called "`ADF-gray-150`" and second "`Flat-Photo 9x13`". Profile name should be short and easily understandable. E.g. First part of name in the example says if automatic document feeder (ADF) is used or flat scanner (Flat).

Example 6.1. scans.xml

```
<?xml version="1.0"?>
<scan_formats>
 <scanner ProductName="*">
  <format id="1" name="ADF-gray-150"
          resolution="150" depth="8"
          feeder="1" autofeed="1"
          duplex="0"
          size_x="8.268" size_y="11.692"
          option="compression=30;" />
  <format id="2" name="Flat-Photo 9x13"
          resolution="300" depth="24"
          feeder="0" autofeed="0"
          duplex="0"
          size_y="4.2" size_x="5.8" />
 </scanner>
</scan_formats>
```

There have to be root tag called `<scan_formats>`. This contains one or more tags called `<scanner>` each describing configuration for one scanner. Common configuration for all scanners is `<scanner ProductName="*">` and each configuration file heve to contain such entry. There can be used name of the scanner instead of asterisk for individual scanner configuration.

Tag `<scanner>` contains list of predefined scanning profiles. Each of them is defined inside separate tag `<format>`.

Table 6.1. scans.xml, tag format

Attribute	Mandatory	Description
id	Y	Identificator of the entry.
name	Y	Name of the profile - user visible
resolution	Y	Resolution in DPI
depth	Y	Bit-depth, can be 1, 8, 24.
threshold	N	Threshold (0-255), valid only for `depth="1"`
option	N	String describing options for used codec.
xfer	N	Transfer protocol - communication between Tahiti and scanner. Only Twain experts should change this value. Possible values: NATIVE, FILE, MEMORY.
xferFormat	N	Transfer format if `xfer="FILE"`. Possible values: `TIFF`, `PICT`, `BMP`, `XBM`, `JFIF`, `FPX`, `TIFFMULTI`, `PNG`, `SPIFF`, `EXIF`
size_x	N	Page size in inches (width).
size_y	N	Page size in inches (height).
feeder	N	Flag is feeder shoould be used. 0 - not used, 1 - use automatic feeder
duplex	N	Flag if use duplex scanning. O - no used, 1 - use duplex scanning
pageFormat	N	Format of scanned page, can be use instead of `size_x`, `size_y`. Available values: `A3`, `A4`, `A5`, `B3`, `B4`, `B5`, `C3`, `C4`, `C5`, `LETTER`, `USLEGAL`. Some scanners do not allow to set `size_x` and `size_y` and only page format can be specified.

Attribute	Mandatory	Description
transformation	N	Transformation function

2. Attributes generation

During scanning process various attributes can be generated and used in created documents. Attribute generation is driven by configuration file scanattrs.xml.

Example 6.2. scanattrs.xml

```
<ScanAttributes>
 <Profile name="All">
  <Fixed id="Scan.Agency" name="Agentura" value="TA"/>
  <Fixed id="Scan.separator" name="separator" value=""/>
  <DocumentType id="Document.type" name="Dokument" shared="0"/>
  <Incremented id="CISLO_JEDNACI" name="CISLO_JEDNACI"
             prefix="" length="4" shared="0" incrOnValue="1"/>
  <Fixed id="HOSP_ROK" name="HOSP_ROK" value=""/>
  <Fixed id="OBDOBI" name="OBDOBI" value=""/>
 </Profile>
</ScanAttributes>
```

Attributes are generated in groups (all attributes from active group are inserted into newly created document). Group is defined in section <Profile>. Every group has it's name and set of attribute generators. During scanning at most one group can be active (user select active group by it's name in Tahiti). Group can contain attribute generators of following types:

- Fixed
- DocumentType
- Incremented

2.1. Fixed

Fixed attribute generator produce constant value for every new document.

Table 6.2. Attributes

Attribute Name	Description
id	Attribute with this id will be added to created document.
name	Name of attribute. This name is displayed in Tahiti.
value	Value of attribute. Can be changed in Tahiti.

2.2. DocumentType

DocumentType attribute generator produce attribute which contains name of document type for every new document. Value can be set in Tahiti where user can select document type from all document types supported in Tahiti.

Table 6.3. Attributes

Attribute Name	Description
id	Attribute with this id will be added to created document.

Attribute Name	Description
name	Name of attribute. This name is displayed in Tahiti.
shared	1-value of this attribute is shared across all groups. 0-value is local for this group.

2.3. Incremented

Incremented attribute generator produce attribute which contains value created from prefix and numerical part for every new document. Numerical part is incremented on given event type. Value is incremented before inserted into document. Last used value is stored on disk for next use.

Table 6.4. Attributes

Attribute Name	Description
id	Attribute with this id will be added to created document.
name	Name of attribute. This name is displayed in Tahiti.
prefix	Prefix of value
length	Length of value
incrOnValue	Event type for increment numerical part of value. Numerical part is incremented when value of attribute Scan.separator is same as given value. Tahiti internally generate following values of attribute Scan.separator: 1-separation page type-1, 2-separation page type-2
shared	1-value of this attribute is shared across all groups. 0-value is local for this group.

It is possible to generate current date, user name as part of prefix. Variables usable in the prefix:

%y year (last 2 digits)

%m month (2 digits)

%d day (2 digits)

%H hours

%M minutes

%u username

```
prefix="cp-%y%m%d" - will generate string with prefix and current date, e.g. cp-080517
```

3. Empty page detection

During scanning process it is posible to detect and reject empty pages. It is very usefull when duplex scan mode is used.

3.1. Attributes

Parameters are set in tahiti.xml and can be overwritten in domain.xml.

Table 6.5. Parameters driving detection of empty page.

Scan.EmptyPage.Soil.Level	Detection of "interesting" pixels (pixels carrying information). Pixel is interesting when its

	intensity differ from average value more then Scan.EmptyPage.Soil.Level. Posible values <0,255>. Default value 15.
Scan.EmptyPage.Soil.Ratio	Factor of filling of page <0,10000>. 0 - no data on page. Page is not empty when detected filling is greater than Scan.EmptyPage.Soil.Ratio. Default value - 90 (at least 0.9 % of page is filled)
Scan.EmptyPage.Side	Size of strip of ignored part of image <0,1000> per mille. Default value - 30 (ignore 3 % from each margin).
Scan.EmptyPage.Type	Type of detection algorithm 2-old, 3-new (recomended).

Chapter 7. Document Assembly

Document Assembly is specialized user interface for batch processing. It is usually used for document identification. It have to be used together with Damis Server backend - description of this server component is in separate documentation.

1. Configuration

This function can be enabled in the `Tahiti.xml`. Main option is called `Damis.Enable` - switch on/off the component.

Table 7.1. Document Assembly Configuration

Configuration	Global	Description
Damis.Enable	Y	Enable/disable component. 0 - disabled, 1 - enabled
Damis.Menu.Revoke.BadContent.Allow	N	Allow to revoke unreadable item and send it back for re-scan. 0 - disabled, 1 - enabled
Damis.Menu.Revoke.UnableToProceed.Allow	N	Allow to revoke documents which use. 0 - disabled, 1 - enabled
Damis.Tree.Sort	N	Sort the tree by default. 0 - disabled, 1 - enabled
Damis.InputPlugins	N	List of input plugins used for first time when document is opened. Detail description is in the following chapter.
Damis.OnReceived.InputPlugins	N	List of plugins used when document is received. Detail description is in the following chapter.
Damis.ExportPlugins	N	List of plugins used to export selected documents.

Note

Global configuration options cannot be overriden in domain specific configuration.

Configuration of connection is stored in file called `damis.xml`. This file can be placed in the domain configuration directory or in the repository.

2. Damis.xml

File contains list of Damis servers. Tahiti can connect to these servers and process stored documents. It is also possible to send new documents from the scanning to them (if allowed).

Example 7.1. Example of Damis.xml

```
<?xml version="1.0" ?>
<DistrScanConfig>
 <Server name="server1" address="10.2.0.2" port="80"
         prefix="/DamisServer/Upload"
         menuScan="Send to Damis"
         uploadUniqueAttribute="FileNumber"
         uploadSplitAttribte="SplitId"
         uploadDomain="scan"
 />
 <Server name="server2" address="10.2.0.2" port="80"
         prefix="" interval="600" simulation="1" />
 <Server name="server3" address="10.2.0.2" port="80"
         prefix="/DamisServer/DamisServlet"
         interval="600"
         autoRequest="1"
 />
</DistrScanConfig>
```

Server is defined in the tag `Server`. Each server have to be identified by the name and address where to connect.

Table 7.2. Damis.xml, tag Server

Attribute	Mandatory	Description
name	Y	Name of the server, used as symbolic name
address	Y	Server address
port	Y	Server port
prefix	N	Prefix where is Damis running on the server. If empty no prefix is used.
interval	N	Interval in seconds to periodicly check if package is valid. Common value is 300 or 600s (10 minutes). 0 - no periodical checks.
simulation	N	Simulation of connection - used for development. 0 - not used, 1 - connection is simulated
autoRequest	N	Automaticly request pending jobs. 0 - off, 1 - on
menuScan	N	Text which will be displayed in the scanning menu allowing to send documents to the given Damis server. If this item is not defined documents cannot be uploaded.
uploadOnlyActive	N	Set if only active version or all versions should be uploaded. N - all versions will be uploaded, Y - only active version will be uploaded
uploadDomain	N	Name of Damis domain used for upload. This have to be name of existing and valid domain.
uploadUniqueAttribute	N	If menuScan is specified. uploadUniqueAttribute or requestUniqueAttribute have to be s defined .This attribute have to be unique for each document set. Documents have to have this attribute.
requestUniqueAttribute	N	If menuScan is specified. uploadUniqueAttribute or requestUniqueAttribute have to be s defined .This attribute have to be unique for each document set. Attribute is issued by server.

Attribute	Mandatory	Description
uploadSplitAttribute	N	This is name of attribute used to split selected batch with documents into smaller packages. Usually uploadUniqueAttribute is used. If this attribute is not specified unique attribute will be issued by server.
blockSize	N	Size of block for data transmition. When set batch is uploaded to server in data blocks of size blockSize, when communication fail Tahiti try to resend last block of data. When doesn't set (attribute is missing) batch is uploaded in one large block. Recomended value > 64000.

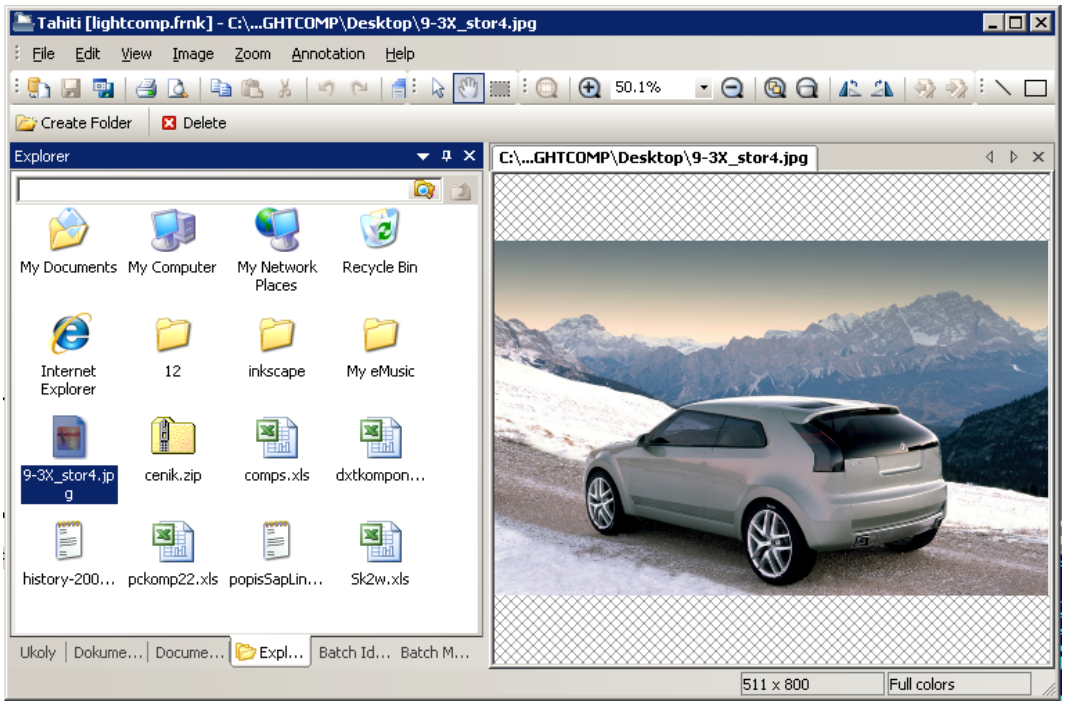
Chapter 8. Components

There is description of several components distributed together with the Tahiti. Components are used to extend Tahiti with new functionality. It can be only small utility or bigger extensions like Explorer integration.

1. Explorer

Explorer is domain specific component. It can be constructed using `contract-id: @lightcomp.com/tahiti/shell-files;1`. It is visible in the list of domain as `Component.ShellFiles`.

Components display embedded Explorer in the one window. It is possible drag&drop files from the explorer to other Tahiti windows or to drop files to the Explorer. There is only one option called `Component.ShellFiles.Directory`. This option allow to specify default directory when Tahiti is started.



Embedded Explorer has one dialog-bar. This bar can be positioned some way as other bars.

2. Update

Update component allows to check if there is new version of the application and start update process. Component internally use internally Invoker to do update. It means Invoker have to be first correctly installed and configured. There are two options in the `Tahiti.xml` which have to be set correctly.

At the first component have to be enabled - have to be listed in the option `SharedComponents.XPCOM.Load`.

Example 8.1. Update - SharedComponents.XPCOM.Load

```
<Item name="SharedComponents.XPCOM.Load" type="string"
   default="
      @lightcomp.com/tahiti/update;1
   ">
  <Description value="Shared XPCOM components."/>
</Item>
```

The second option is called `Component.Update.InvokerFile` and is also placed in the `Tahiti.xml`. It contains name of configuration file for the Invoker. Tahiti is using this file same way as `Invoker.exe`. This file is typically called `tahiti-update.xml`.

Component also allows to start update process from the server. It can be done by sending Action Object (.taobj) to the Tahiti. Such object can start check if there is new version. Action object can also contain several options which can force Tahiti to use different XML file for the check. Description of the Action Object and attributes is part of the DEP2 Specification.

Chapter 9. Input Plug-ins

Input Plug-ins are components which allows preprocessing of newly added files. These Plug-ins are applied on the locally added files. They can be also applied on files received from the Damis.

Commonly used input plug-ins:

Basic Filter	Limits types of added files as well as their size.
Photo Filter	Allows to downscale, crop and limit size of photos. Filter also make basic format conversion (e.g. BMP->TIFF).
Tiff Splitter	Split multi-page TIFF files into several one page files.
Specialized Filters	Various filters for conversion from proprietary formats to HTML, PDF, etc...

Input Plug-ins are applied on several events:

- new file is added to the document
- file is added/received to the Document Assembly Tree
- file in the Document Assembly Tree is opened for first time

1. Configuration

Filters are configured in the `Tahiti.xml`. There are separate lists of filters used for each event. List contains comma separated names of filters. Filters are applied in the order as they are in the configuration.

Example 9.1. Example of Input Plug-ins Configuration

```
<Item name="DocTree.AddFile.InputPlugins" type="string"
   default="@lightcomp.com/tahiti.input_plugin.tiffsplitter;1,
            @lightcomp.com/tahiti.input_plugin.convertavn;1,
            @lightcomp.com/tahiti.input_plugin.photofilter;1,
            @lightcomp.com/tahiti.input_plugin.filter;1">
 <Description value=""/>
</Item>
```

This configuration will 4 filters on every newly added file to the tree. Possible Input Plug-ins configurations:

Table 9.1. Configurations of Input Plug-ins

Configuration Name	Context	Usage
DocTree.AddFile.InputPlugins	`DocTree`	Filters used when new file is added to the tree.
Damis.InputPlugins	`Damis`	Filters used when file is opened for first time.
Damis.OnReceived.InputPlugins	`DamisOnReceived`	Filters used when file is received from the server.

There is common configuration for all filters as well as specialized configuration for each filter usage - called context. Each filter configuration contains word `Default`. Such configuration is used for

all contexts. Context specific configuration has instead of name `Default` name of the context, e.g.: `InputPlugin.Filter.DocTree.Include.1`, can be used to limit used files in the context `DocTree`.

1.1. Damis.InputPlugins

Filters are applied when user open file for first time. There have to be attribute `Damis.InputFilter` with 2nd bit on (e.g.: Damis.InputFilter=2).

1.2. Damis.OnReceived.InputPlugins

Filters are applied on received documents/files from the Damis server. There have to be attribute `Damis.InputFilter` with lowest bit on. Filters are applied in separate thread and have to be without user interface. Tiff Splitter is usually called here.

2. Basic Filter

Basic Filter is used for limiting size of files and also types. It is possible to limit maximum file size or warn user when file is greater then recommended size.

Filter name: `@lightcomp.com/tahiti.input_plugin.filter;1`

Table 9.2. Basic filter configuration

Configuration	Description
InputPlugin.Filter.Default.Include.1	List of allowed mime-types. If this list is empty all types are allowed. Files with different types then in the list are not allowed.
InputPlugin.Filter.Default.Include.2	List of recommended mime-types. If this list is non empty and mime-type of the file is not in the list warning is displayed.
InputPlugin.Filter.Default.Exclude.1	List of forbidden types. If mime-type is in the list file is forbidden.
InputPlugin.Filter.Default.Exclude.2	List of not recommended mime-types. If mime-type is in the list user have to confirm that file is correct.
InputPlugin.Filter.Default.MaxSize.1	Maximum file size in bytes. Bigger files than this limit are not allowed. When this value is 0 no limit is applied.
InputPlugin.Filter.Default.MaxSize.2	Maximum recommended size in bytes. User have to explicitly confirm bigger files than the limit.

3. Photo Filter

Photo Filter is visual component used to achieve smaller files. Component also provides automatic conversion of BMP files to optimized TIFF files.

Filter name: `@lightcomp.com/tahiti.input_plugin.photofilter;1`

Supported operations:

- downscale
- crop
- save as JPEG with lower quality

Table 9.3. Photo filter configuration

Configuration	Description
InputPlugin.PhotoFilter.Default.FileSize	Minimal file size to activate this filter.
InputPlugin.PhotoFilter.Default.OptimalX	Optimal image size in X-coordinate. 640 pixels is default size.
InputPlugin.PhotoFilter.Default.OptimalY	Optimal image size in Y-coordinate. 480 pixels is default size.
InputPlugin.PhotoFilter.Default.Size.Warn	Maximal acceptable file size. File can pass through the filter but warning is displayed.
InputPlugin.PhotoFilter.Default.Size.Ok	Maximal recommended size.

4. Tiff Splitter

Tiff Splitter is used to split multi-page Tiff file into several one-page TIFF files. This component is without user interface and has no configuration.

Filter name: `@lightcomp.com/tahiti.input_plugin.tiffsplitter;1`

5. AVN Filter

AVN filter is used to convert output from software called AVN to HTML. Filter automatically detects file encoding and format. This filter has no user interface and no configuration.

Filter name: `@lightcomp.com/tahiti.input_plugin.convertavn;1`

6. ChangeMimetype filter

ChangeMimetype filter is used to change Mimetype of outlook messages (files .msg). Filter is activated when file with mimetype application/msoutlook or application/x-msoutlook is inserted. Mimetype is changed to value obtained from configuration from variable named "Views.Mail.Outlook.MimeType"

Filter name: `@lightcomp.com/tahiti/change_mimetype;1`

Chapter 10. Add-ons for Applications

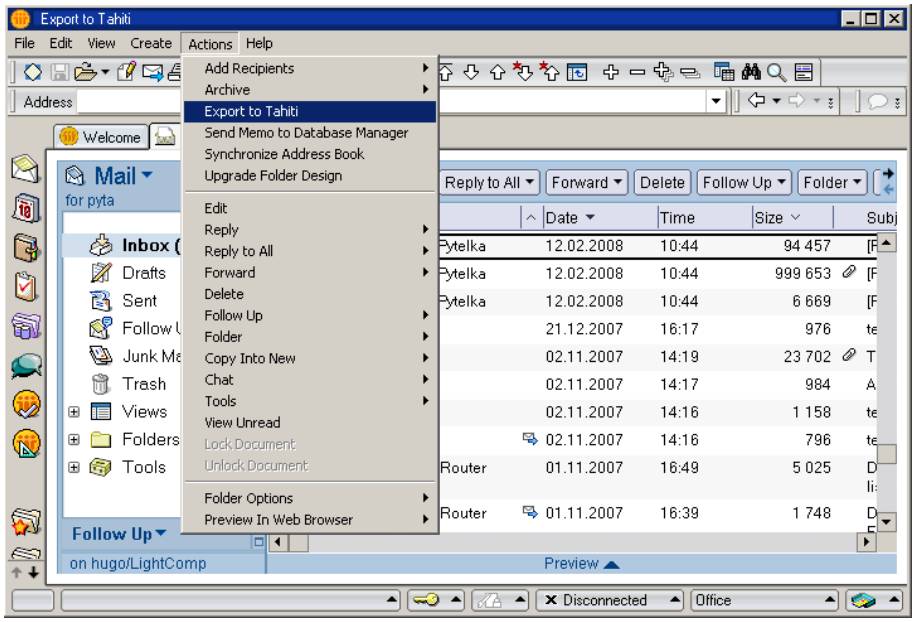
Add-ons are components for external Applications which allows saving documents/data to the Tahiti. There are several components for various applications and others can be created using Tahiti COM objects. Detail description is part of the Tahiti SDK.

1. Internet Explorer

There is Internet Explorer extension called `IEHelper.dll`. This file have to be registered to be able to save actual page to the Tahiti.

2. Lotus Notes

It is possible to save email from Lotus Notes directly to the Tahiti. There is new Tahiti specific item in the menu Actions.



This add-on have to be installed in the Lotus Notes mail template on the server side. Usually file is called `mailX.ntf` (e.g. `mail7.ntf`). There are just two parts to do this. The first is form and the second is LotusScript that is called from an action. Steps to install add-ons:

1. Create new form called MimeConverter

2. Add new Agent "Export to Tahiti"

3. Distribute template to the clients

This add-on requires Tahiti 4.5.0.8 or later. Library `OfficeLink3Proxy.dll` have to be registered during installation.

2.1. Form MimeConverter

This form has just two fields. The first is `MimeRichTextField`. This should be set to store contents as HTML and MIME and is a `RichTextField`, which is also editable. The second field is called `HtmlText` and is also an editable `RichText` field. Let it be known that this is a standard Lotus Notes Richtext field. The form should be called `MimeConvert` and created in the mail file or mail file template.

This form is used internally by newly created Agent.

2.2. Agent "Export to Tahiti"

The LotusScript code is part of the Tahiti SDK, file `doc\addons\lotus-notes\script.txt`. This code should be copied into an agent. Run it from the action menu and on all selected documents.

3. Microsoft Office

There are extensions for Excel, Outlook and Word. Each extension have to be registered and it will automatically appear in the File menu of each application.

Table 10.1. List of Microsoft Office Add-ons

Application	Dll
Excel	`ExcelAddin.dll`
Outlook	`OutlookAddin.dll`
Word	`WordAddin.dll`

3.1. Troubleshooting

It is possible to log any event in the extension and use the log to resolve potential problems. Logging can be set in the registry - if there is valid path to the log file component will log all events.

Path: `HKCU\Software\LightComp\Tahiti\5.0`

Table 10.2. List of Microsoft Office Add-ons Registry keys

Key	Description
ExcelAddinLog	Excel specific key, string value - path to the log file including file name.
OutlookAddinLog	Outlook specific key, string value - path to the log file including file name.
WordAddinLog	Word specific key, string value - path to the log file including file name.

4. OpenOffice

There is extension (`TahitiOOExt.oxt`) for Writer, Calc, Draw and Impress. This extension allows to save file directly to the Tahiti. Extension is common for all applications and can be installed automaticly during Tahiti installation or manualy. Use OpenOffice extension dialog to install this extension manualy. Check OpenOffice documentation for possible ways of installing extensions.

4.1. Registration

During installation from MSI if OpenOffice is installed extension will be automaticly registered. There is utility `RegisterOOExt`.exe which simplifies registration of the extension. File will register extension for all users (as shared extension) and can be used only when there is no running OpenOffice.

Note

`RegisterOOExt` calls internaly `unopkg.com` which is part of the OpenOffice.org installation. Extension can be registered only if there is no running OpenOffice.

`RegisterOOExt` returns 0 if registration succeeded or OpenOffice installation was not found. Utility returns error code 1 if registration of the extension fail. It means that OpenOffice is running or insufficient rights.