# Kofax Communication Server

## TWS Technical Manual

Version: 10.3.0

Date: 2019-12-13

KOFAX

# Table of Contents

Chapter 1

# Preface

TCOSS Web Services (TWS) for Kofax Communication Server (KCS), also called Kofax Communication Server Connector provides a web services interface for TCOSS and TC/Archive. TWS is not an end-user application. It is primarily intended for third party developers, workflow and web designers.

**Third party developers** can use TWS to access all features of TCOSS and TC/Archive via Web Services. E.g., it is now possible to search for messages in the archive or to check the state of messages in the out box. These features are currently not possible with TC/SOAP. Web services are supported by various developer tools like Microsoft Visual Studio .net. TWS provides the WSDL (Web Services Definition Language) files required to generate a source code for using the web service.

TCOSS and TC/Archive can be integrated into a service-oriented architecture (SOA) by using TWS. Workflow systems (e.g. Inubit integration server) typically support web services as a generic interface to external components. This means that a **workflow designer** can use TWS instead or in addition to TC/SOAP.

**Web designers** can integrate a simple real-time output from TCOSS or TC/Archive into any web page. In that case, a function will be executed according to a specified hyperlink. The result of structured data can be converted into a well formatted HTML page by using style sheets. Some TWS functions convert the content into binary format (e.g., PDF) which can be displayed on the client using the appropriate application. Some samples how to use this integration are shown in the TWS web portal which is installed as part of TWS.

When using TWS, the user credentials (KCS user ID and password) can be either defined by configuration or they have to be provided by the client using HTTP basic authentication. In the second case, impersonation is used. No user ID or password has to be configured for TWS. Each user gets the permissions according to his user profile. The permissions are checked on the TCOSS and TC/ARCHIVE server.

The web service interface is based on the Simple Object Access Protocol (SOAP) 1.1. It uses the “document/literal” style SOAP binding (see *Conformance to Standards*).

**Important The Kofax Communication Server and its components formerly used the name TOPCALL. Some screen shots and texts in this manual may still use the former name.**

## Feature Matrix

TWS provides a similar set of features like TFC and TC/SOAP. The table below provides a rough overview about the supported features in the 3 applications.

| Features | TFC | TC/SOAP | TWS |
|---|---|---|---|
| **Interface transport, security** | | | |
| COM interface | # | - | - |
| Web interface (call functions via URL reference) | - | - | # |
| Web Service Interface (SOAP) | - | # | # |
| HTTPS (HTTP over SSL) connection | - | # | # |
| HTTP basic authentication | - | - | # |
| Support impersonation (no user ID / password has to be configured) | - | - | # |
| **Send messages** | | | |
| Convert message using customer specific XSLT transformations before sending | - | # | # |
| Send messages in XML format | - | # | # |
| Real-time response from server when posting messages | # | - | # |
| Document conversion when sending messages (e.g., convert Word attachments to image) | - | # | # |
| Build-in web form for send-message tests/troubleshooting | - | - | # |
| **Receive messages** | | | |
| Convert received messages using customer specific XSLT transformations | - | # | # |
| Receive messages in XML format | - | # | # |
| Call a Web Service when receiving messages (external application is called) | - | # | - |
| Get message via Web Service (polling KCS queue) | #1) | - | # |
| Document conversion when receiving messages (e.g., convert Word attachments to image) | - | - | # |
| **UM functions** | | | |
| Show In box of user | # | - | #2) |
| Show Out box of user | # | - | #2) |
| Open messages from In or Out box | # | - | #2) |
| Search messages in Archive | # | - | # |
| Create and change address book entries | # | - | #2) |
| **Service administration functions** | | | |
| Support TC/XML Dirsync message | - | # | - |
| User profiles (read/write/modify/delete) | # | - | #2) |
| Service (read/write/modify/delete) | #1) | - | #2) |
| Change server date/time | #1) | - | # |
| Get line state | #1) | - | # |

| Features | TFC | TC/SOAP | TWS |
|---|---|---|---|
| Other maintenance functions | # | - | # |
| **Performance, architecture** | | | |
| Support of concurrent calls | # | # | # |
| Real-time calls (without temp-file) to server | # | - | # |
| **Supported Application Extensions** | | | |
| Fax Connector for KFS (Kofax Front Office Server) | - | - | # |
| Advanced Kofax Capture Interconnection | - | - | # |
| Kofax Capture Import Connector Fax | - | - | # |

1) Not possible with native TFC only using undocumented TCSI calls.

2) No simple workflow method available, knowledge of TCSI structure required.

Chapter 2

# Functional Definition

TWS provides a toolset for system integrators giving full access to KCS functionality via open standards. They are installed as part of the Kofax Communication Server and provide the following three groups of functions. Each group is described by a WSDL (web service definition language) file.

- TCOSS functions (described in tcoss.wsdl)
- TCARCHIVE functions (described in tcarchive.wsdl)
- Micro Workflow functions (described in tsl.wsdl and tsl1.wsdl)

The TCOSS functions provide full access to the connected TCOSS server. The TCARCHIVE functions do the same for the TC/Archive server. In both cases a web-service call is translated using a generic translation into a TCSI call (KCS Client Server Interface). Therefore, a complete description of all possible functions can be found in the TCSI manual.

The micro workflow functions are intended to define simple web service calls (that may even be customer-specific). Such a workflow function may include one or more TCOSS or TCARCHIVE function calls. This version of TWS contains the following functions:

- Open a message from Archive and return its content as XML document
- Open a message from Archive and return its content converted into PDF
- Post a simple message via web service.
- Get messages from TCOSS

The file *tsl.wsdl* contains the interface of all functions. File *tsl1.wsdl* is a subset of *tsl.wsdl* containing only functions that do not require the TCSI schema. Therefore, *tsl1.wsdl* is much more compact than *tsl.wsdl*.

**Important** TWS does not need a separate web server application (like IIS, apache, etc.). If any web server runs on the machine together with TWS, make sure that it is using a different port. The default port number of TWS is 25082.

## Time Zone Support

In general all fields of type “dateTime” in web service requests and responses are in TCOSS server time. Depending on configuration TCOSS server time can be UTC, UTC with offset, or a local time zone. The recommended configuration is to have TCOSS running on UTC.

The following table shows how “dateTime” fields in Requests are handled:

| “dateTime” Format | TCOSS on UTC | TCOSS on UTC + offset | TCOSS on local time |
|---|---|---|---|
| Without time zone, e.g. 2010-01-21T10:25:45 | No conversion, taken as UTC | No conversion, taken as UTC + offset | No conversion, taken as local time |

| "dateTime" Format | TCOSS on UTC | TCOSS on UTC + offset | TCOSS on local time |
|---|---|---|---|
| UTC with following "Z", e.g. 2010-01-21T09:25:45Z | No conversion, is UTC | Converted into UTC + offset | No conversion, taken as local time |
| With time zone, e.g. 2010-01-21T10:25:45+01:00 | Converted into UTC | Converted into UTC + offset | No conversion, taken as local time |

The following table shows how "dateTime" fields are given in Responses:

| "dateTime" Format | TCOSS on UTC | TCOSS on UTC + offset | TCOSS on local time |
|---|---|---|---|
| Without time zone, e.g. 2010-01-21T10:25:45 | Is UTC | Is UTC + offset | Is local time |

# Multiple Instances for xSPs

Currently, TWS can access only one TCOSS or TCARCHIVE server instance. Multiple TCOSS / TCARCHIVE instances are possible with multiple TWS instances. Although such a configuration scenario does not exist in the setup at present, it can be implemented on demand. Please contact Kofax staff if you are interested.

# Licenses

TWS requires a TWS license. TWS licenses are counted by workstation. You can contact Kofax to request a free temporary license.

Some TWS functions convert messages into PDF (Portable Document Format). Even in such a case no additional PDF license is required.

# Security Features

This section describes the security features.

## Client User Authentication

The application provides the following modes of authentication:

- HTTP basic authentication (impersonation)
- Predefined KCS user ID and password.

If no user ID/password is configured, each request from the client must contain a valid KCS user ID and password. They are used to log in to the TCOSS or TCARCHIVE server. If no or invalid user credentials are provided, the request will be rejected with an appropriate http response.

If a user ID and password are configured, these user credentials will be used for all calls. The password is stored in an encrypted form in the configuration file.

Whenever a TCOSS or TC/Archive function is called the TCOSS or TC/Archive server checks if the current user has enough permission to perform the requested action. As a general rule, users that calls a function in TWS (via web interface or web service) needs the same permission in the TCOSS user profile as if the same action would be done with TCfW.

## HTTP over SSL (HTTPS)

The application can be configured to support either standard HTTP or HTTPS. If you use the HTTP basic authentication for identifying the clients, it is strongly recommended to use https. Otherwise, passwords might be hacked by sniffing the network traffic.

If HTTPS is used without certificate, the connection between client and server will still be encrypted, but the client has no guarantee to be connected with the intended server which means the connection will not be protected against “man in the middle” and “IP spoofing” attacks.

There is no error message (e.g. event log entry) if the certificate will expire or is already expired. But the clients that access the server will get an appropriate warning.

The HTTPS implementation is based on the OpenSSL Project, an Open Source toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1, TLS v1.1 and TLS v1.2) protocols. OpenSSL version 1.0.2o from 27-Mar-2018 is used (see http://www.openssl.org).

### Disable TLS 1.0

To disable TLS 1.0, do the following:

1. Open Create_Config.xslt. Default installation path is
   `TWS\00\xcd\Create_Config.xslt`
2. Add the following code between the `Threads` and `EventRateLimiter` elements as shown below.

```
<OpenSsl>

            <ContextOptions>117440512</ContextOptions> <!-- 0x07000000: Disable
 SSL2, SSL3, TLS1.0 -->

        </OpenSsl>
```

3. Save and close Create_Config.xslt.

## Allow Local Connections Only

TWS may be configured to accept only local connections from the same PC. This may be useful if the web service interface is accessed only by locally installed applications, so there is no need to listen to connections from outside.

Chapter 3

# Installation

This section describes the installation of TWS.

## Prerequisites

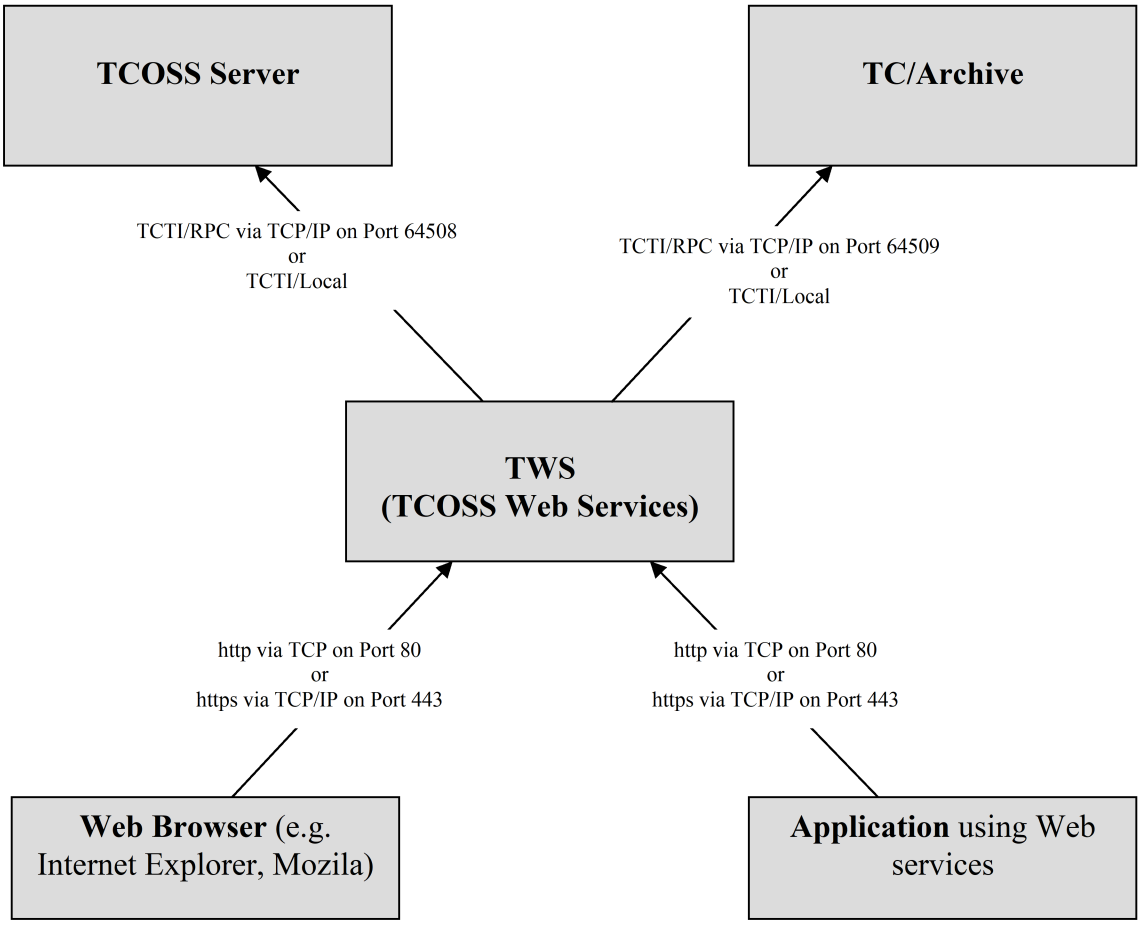
Same as defined for Kofax Communication Server containing TWS.

The product is supported with the following TCOSS and TC/Archive versions which have been released in KCS 7.59.06:

- TCOSS Server version must be 7.88.00 or higher
- TC/Archive Server version must be 2.11.00 or higher

### Network Requirements

The image below gives you an overview about the involved network connections. Each connection is indicated with an arrow from the client to the server. The text on the arrow described the typically used protocol.




The network connection between TWS and TCOSS or TC/Archive uses TCTI. This means that the requirements are the same as with TCfW or TC/Web. If both applications are running on the same computer, it is recommended to use the TCTI Local transport. Otherwise, TCTI via TCP/IP should be used.

The network connection between the web browser and service client to TWS must allow TCP connections (from the client to TWS) with the configured port. The default port is 25082.

There are no special bandwidth and latency requirements for the network connections unless the application using the web services has some real-time constraints.

## How to Check TCP Port Usage

TWS requires a free TCP port to run an HTTP server. If you are not sure whether the certain ports, e.g., 80 for HTTP and 443 for HTTPS are already in use by another application, open a “cmd” window and use the “netstat” command to get an overview of TCP connections:

```
C:\>netstat -a -b -n -p TCP
Active Connections
  Proto  Local Address          Foreign Address        State           PID
```

```
  TCP    0.0.0.0:80             0.0.0.0:0              LISTENING       1156
  [Skype.exe]
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING       1196
  RpcSs
  [svchost.exe]
  TCP    0.0.0.0:443            0.0.0.0:0              LISTENING       1156
  [Skype.exe]
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING       4
  [System]
  TCP    0.0.0.0:1025           0.0.0.0:0              LISTENING       772
  [lsass.exe]
  TCP    0.0.0.0:62463          0.0.0.0:0              LISTENING       1156
  [Skype.exe]
```

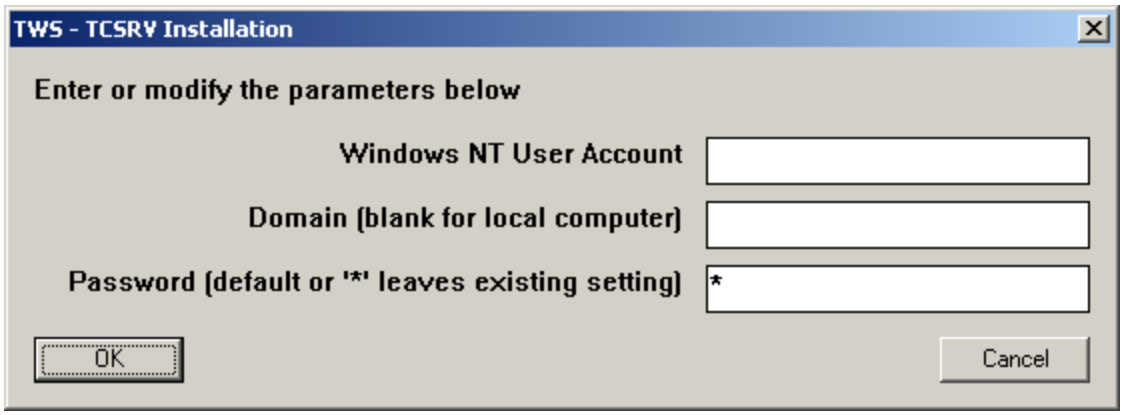
In the above example ports 80 and 443 are already in use by the “Skype” application. If TWS is running and listens on port 80 it looks like this:

```
C:\>netstat -a -b -n -p TCP
Active Connections
  Proto  Local Address          Foreign Address        State           PID
  TCP    0.0.0.0:80             0.0.0.0:0              LISTENING       3888
  [TnAppl1.exe]
```

# Installation Procedure

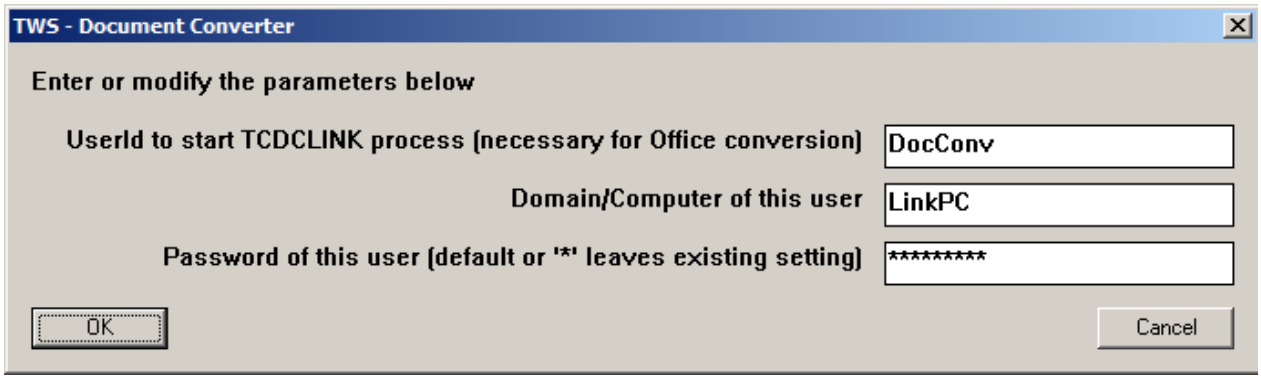
TWS is delivered either as a part of KCS or as a standalone installer. This chapter describes the installation using KCS setup.

Start the setup. The application can be found in the group Common > Application Interfaces and Services. Select “TWS”. The installation procedure prompts for a Windows user account as shown in the screen shot below:



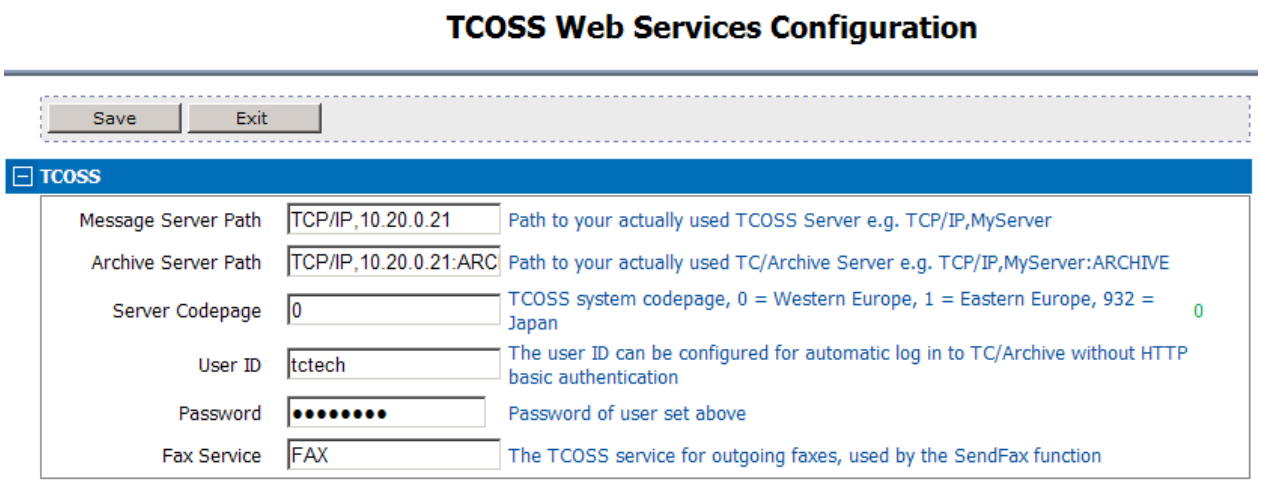

If you enter Windows user account, TWS will be launched with that permission. But typically, no special user account is needed. In that case, TWS will use the TCSRV user account (by default, this will be the System account).

Document Converter is automatically installed along with TWS. To use Microsoft Office applications it is necessary to configure a user for the Document Converter (TCDCLINK) process. The same user has to be configured with the Windows tool “Dcomcnfg” (only necessary on Windows Server 2008 and later).




- **UserId to start TCDCLINK process (necessary for Office conversion)** (registry TCDCLINK\UserId, default: empty) – user to start the TCDCLINK process with
- **Domain/Computer of this user** (registry TCDCLINK\Domain, default: empty) – domain/computer of above user
- **Password of this user (default or "*" leaves existing setting)** (registry TCDCLINK\Password, default: empty) – password of above user

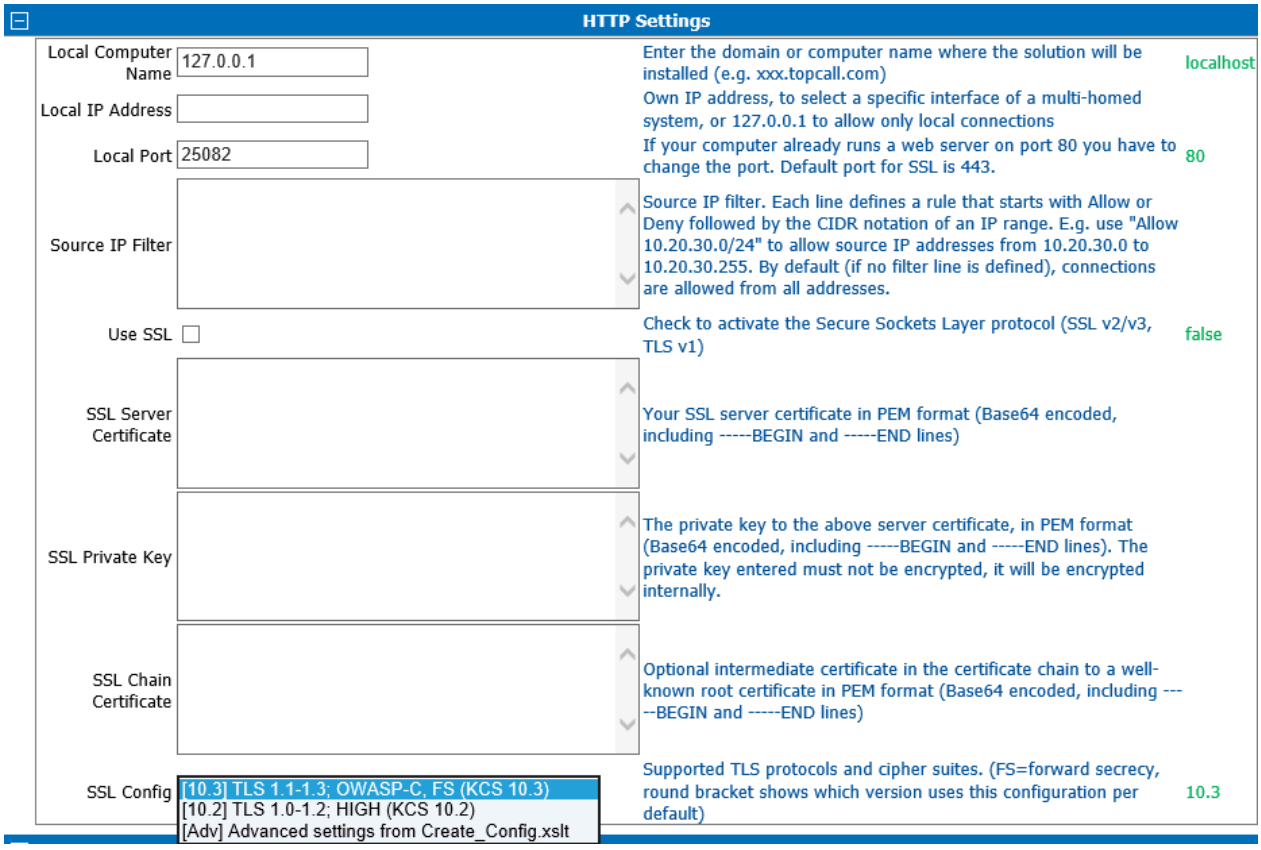
Then, a configuration page is shown where installation specific data have to be entered. See screen shots below:




A more detailed description of the parameters in the TCOSS tab can be found below:

| Value | Description |
|---|---|
| Message Server Path | This value should be set to the TCTI Path to your TCOSS Server. If you use only the archive access functions, no access to the TCOSS server will be made and therefore this configuration parameter will not be used. |
| Archive Server Path | This value should be set to the TCTI Path to your TC/Archive Server. |
| Server Codepage | TCOSS system codepage, e.g. 0 = Western Europe, 1 = Eastern Europe, 932 = Japan. This codepage is used for both TCOSS message and TC/Archive server. |

| Value | Description |
|---|---|
| User ID | Default calls that access the TCOSS or TC/Archive server request http basic authentication information from the client. Basic authentication is supported by web browser and it is recommended to be used. If you want to use a predefined KCS user ID/ password (without HTTP authentication), it has to be configured using this configuration parameter. |
| Password | If TcUserId is not empty, its password has to be configured here. The password is stored in an encrypted form. The current password is not shown on the user interface. |
| Fax Service | This value specifies the TCOSS service which is used to address outgoing faxes submitted by the SendFax function. |

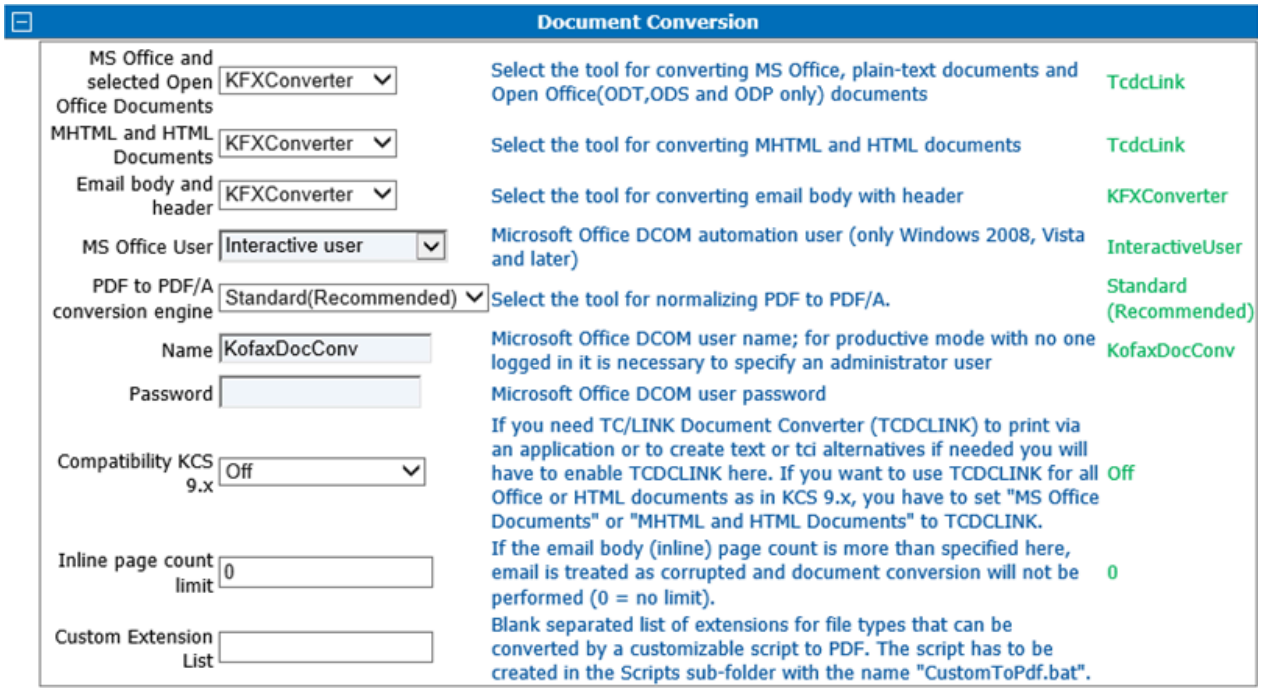



A more detailed description of the parameters in the HTTP Settings tab can be found below:

| Value | Description |
|---|---|
| Local Computer Name | This value specifies the computer name where TWS is installed. It is used to construct the location attribute of the web service in the WSDL files. **Example**: test.kofax.com |
| Local IP Address | Optional own IP address, to select a specific interface of a multi-homed system, or 127.0.0.1 (localhost) to allow only local connections. Enter only addresses here in “1.2.3.4” notation, not host names. |

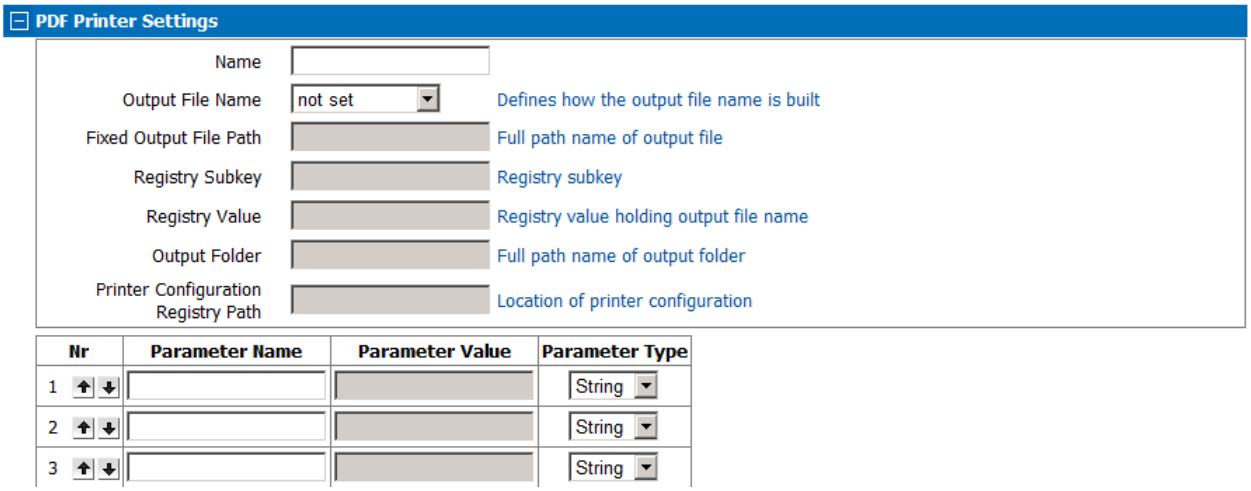
| Value | Description |
|---|---|
| Local Port | The value can be used to specify an alternative port for access of Web Service and maintenance functions. An alternative Port will be required if any other application (e.g. web server) uses port 80 on your machine. The default value is 25082. |
| Source IP Filter | IP filter lines so that connections can be restricted to specific IP address ranges. By default (if no filter line is defined), connections are allowed from all addresses. For more details, refer section Blacklist / white-list support. |
| Use SSL | Check this box to use https instead of http. If using https change also the Local Port from 80 to 443. |
| SSL Server Certificate | Your SSL server certificate in Base64 encoding (including -----BEGIN and -----END lines) |
| SSL Private Key | The RSA private key to the above server certificate, in the same format. This field is stored in an encrypted form. An already stored key is shown as "********" on the user interface. |
| SSL Chain Certificate | Optional intermediate certificate in the certificate chain to a well-known root certificate. This field can also hold the root certificate which was used for signing the server certificate in case it is not a well-known root certificate, e.g. a test root certificate or a root certificate of a private certification authority. |
| SSL Config | For SSL security configuration, select the of the following:<br>• **[10.3] TLS 1.1-1.3; OWASP-C, FS**: Supports TLS 1.1, 1.2 and 1.3. Also, support OWASP Cipher String 'C' and forward secrecy. This is default configuration.<br>• **[10.2] TLS 1.0-1.3; HIGH**: Supports TLS 1.0, 1.1, and 1.2. This configuration (including cipher list) provides the behavior of KCS 10.2.0 as good as possible.<br>• **[Adv] Advanced settings from Create_Config.xslt**: This option is for advanced users or can be used for troubleshooting. The configuration is derived from the `AdvancedOpenSslConf` variable in Create_Config.xslt.<br>`<xs:variable name="AdvancedOpenSslConf"> <!-- OpenSsl configured`<br>`<OpenSsl>`<br>`<ContextOptions>121634816</ContextOptions> <!-- 7400000:`<br>`<CipherList>OWASP-C</CipherList>`<br>`<Flags>0</Flags> <`<br>`</OpenSsl>`<br>`</xs:variable>` |


The "File Types Converted via TCIMGIO" tab allows to define up to 20 extensions and assign them to the predefined file types. For more information, please refer to the KCS Document Conversion Technical Manual.

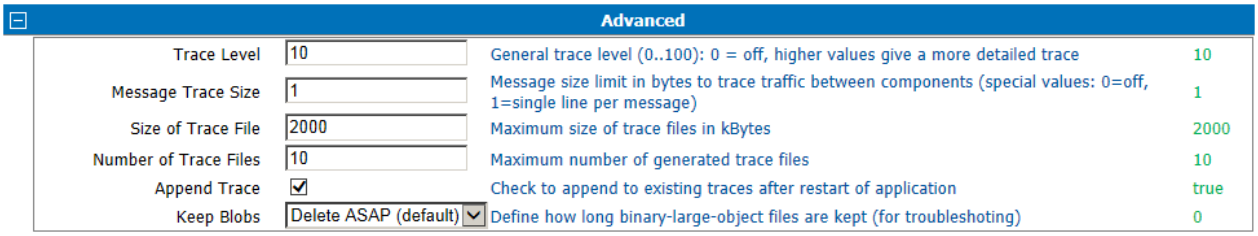



| Value | Description |
|---|---|
| MS Office and selected Open Office Documents | Select the tool for converting MS Office, plain-text and Open Office (ODT, ODS and ODP only) documents. The following options are available: KFXConverter, Microsoft Office, OpenOffice.org, TcdcLink, or <disabled>.<br>**Note** For converting Microsoft Office documents using OpenOffice.org, install the OpenOffice.org extension. |
| MHTML and HTML Documents | Select the tool for converting MHTML and HTML documents. Same options as above. |
| Email body and header | Select the tool to use for converting email body and message header to PDF body: KFXConverter or Microsoft Office.<br>Default: KFXConverter |
| MS Office User<br>Name<br>Password | Select a Windows user account for Microsoft Office DCOM automation (necessary for document conversion via Microsoft Office).<br>• "Interactive user" means that the user must be logged in on the TWS computer. During document conversion, you can observe Office windows opening and closing. This mode is useful for advanced troubleshooting.<br>• "Current configuration" means that Office applications should start with the user account specified in DCOM Config tool.<br>• "This user:" allows you to type a name and password directly.<br>Type a Microsoft Office DCOM user name and password. |
| PDF to PDF/A conversion engine | Select PDF to PDF/A conversion engine for normalizing PDF documents to PDF/A format. It is recommended to use the **Standard(Recommended)** option for document normalization. Default is **Standard(Recommended)**. |

| Value | Description |
|---|---|
| Compatibility KCS 9.x | If you need TC/LINK Document Converter (TCDCLINK) to print via an application or to create text or tci alternatives – you will have to enable TCDCLINK here. If you want to use TCDCLINK for all Office or HTML documents as in KCS 9.x, you have to set "MS Office Documents" or "MHTML and HTML Documents" to TCDCLINK. |
| Inline page count limit | For converting an email body (inline) to PDF, set the maximum limit of inline page count. If the inline page count for an email is more than the specified limit, KFXConverter treats it as corrupt and does not perform the document conversion. If this limit is set to 0, KFXConverter does not treats the emails as corrupt and performs the document conversion.<br>**Note**<br>• This option is only enabled when **KFXConverter** is selected for **Email body with header** option.<br>• In Kofax Capture Connector, the **Include message header of email** option must be selected for **Body with Message Header**. |
| Custom Extension List | Blank separated list of extensions for file types that can be converted by a customizable script to PDF. The script has to be created in the Scripts sub-folder with the name "CustomToPdf.bat". |



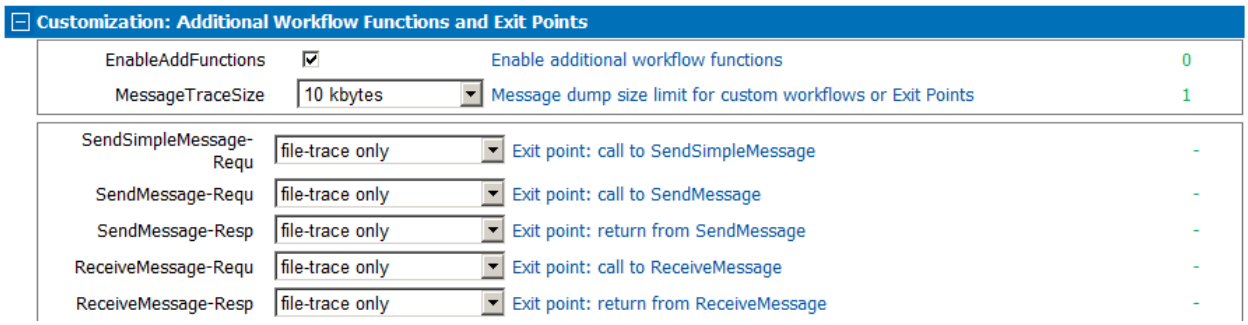

This sections defines parameters for PDF conversion. In the 2nd part it allows to define up to 10 printer parameters and assign values and types to them. These parameters are currently only useful for configuration of the Adobe PDFWriter. For more information PDF conversion configuration values, please refer to the *KCS Document Conversion Technical Manual*.



A more detailed description of the parameters in the Advanced tab can be found below:

| Value | Description |
|---|---|
| Trace Level | General trace level, 0 = off, higher values give a more detailed trace. Please refer to chapter *Troubleshooting* at the end of this manual for information about trace levels and interpreting the traces written by TWS. |
| Message Trace Size | Message size limit in bytes to trace traffic between components |
| Size of Trace File | Maximum size of trace files in kBytes |
| Number of Trace Files | Maximum number of generated trace files. |
| Append Trace | Check to append to exiting Traces after restart of application |
| Keep Blobs | Use the default option unless Kofax Support tells you otherwise. |

Change the values and then click Save to save the modifications. Click Exit to leave the configuration tool.




The customization section may be used to implement new web service calls or to modify some existing function at some exit points. Please refer to chapter *Customization* for detailed description.

If you want to change these settings after installation, the batch file configure.bat the installation directory must be called. KCS setup creates an entry to configuration.bat in the start menu (Kofax\Fax Connector\ KCS Configuration Utility (TWS)). The application must be restarted in order to use the modified values.

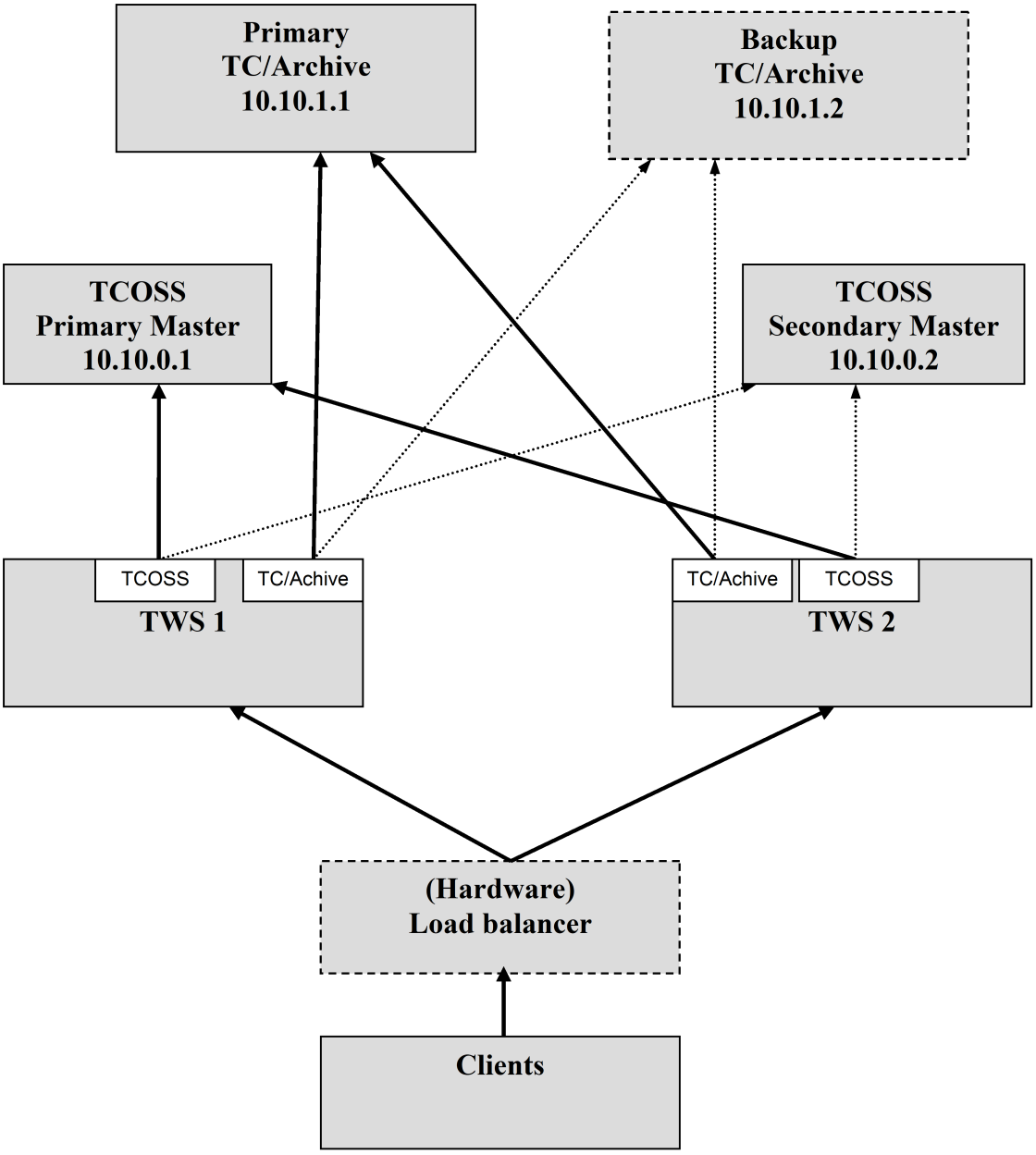
## Installing OpenOffice.org Extension

For converting Microsoft Office documents using OpenOffice.org, you must install the OpenOffice.org extension. This component should be installed on the same computer where TWS is installed. Do the following:

1. Install a supported version (currently certified version OpenOffice.org 3.2) of OpenOffice.org.
2. To install the extensions, browse to the \export\kcs\OpenOfficeExtension folder in KCS setup directory and run Install.bat.

# Fault Tolerant Systems

If you need fault tolerance, TWS must be installed on two different computers. A typical example for using a Tandem TCOSS server is shown in the picture below:




In the example above the path to TCOSS server has to be set to “TCP/IP,10.10.0.1|TCP/IP,10.10.0.2”. The path to the archive server has to be set to “TCP/IP,10.10.1.1:ARCHIVE|TCP/IP,10.10.1.2:ARCHIVE”. The clients can access both TWS instances.

**Hint: DO NOT USE BLANKS between the two addresses.**

A hardware Load Balancer should be installed between the TWS servers and the client network. The Load Balancer has a single external IP address, and routes all requests for a web server to one of the TWS servers.

If one of the TWS servers stops functioning properly, then all requests will be directed to a functioning server. Any users that were using TWS when a server stops working will not get a responds to its last request. But they will be able to continue to work after the request is retransmitted.

The BigIP product from F5 networks (http://www.bigip.com) is an example of a hardware load balancing product.

Alternatively, the client application itself can do a failover to TWS2, if it recognizes that web service requests to TWS1 do not get any response.

# Installation with HTTPS

This chapter describes the whole procedure that is required if you want to use a secure connection to TWS (using https).

## Step 1: Create a Private Key and a Certification Request

Open a command prompt and go to the C:\Topcall\Shared directory. The following files will be used:

- Openssl.exe
- Openssl.cnf

Call the openSSL tool to create a private key and a certificate request. Enter the appropriate values according to your environment (shown bold). Make sure that the “Common Name” exactly matches the host or domain in the client URL (e.g. “https://*host*/file/index.html”). Normally, this will be a full qualified domain name, e.g. *host.company.com*. If it does not match the browser will display a warning like “The name on the security certificate is invalid or does not match the name of the site. Do you want to proceed?” and a SOAP call via HTTPS will probably fail.

```
C:\topcall\SHARED>set OPENSSL_CONF=openssl.cnf
C:\topcall\SHARED>openssl req -config openssl.cnf -new -nodes -keyout private.pem -out
 request.pem
-days 3650


Loading 'screen' into random state - done
Generating a 1024 bit RSA private key
.........++++++
.....++++++
writing new private key to 'private.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AT]:AT
State or Province Name (full name) [Austria]:Austria
Locality Name (eg, city) [Vienna]:Vienna
```

```
Organization Name (eg, company) [Dogbert Engineering Ltd]:Kofax
Organizational Unit Name (eg, section) []:
Common Name (host or domain as in client URL) []:VMSB-TCOSS8WEBS
```

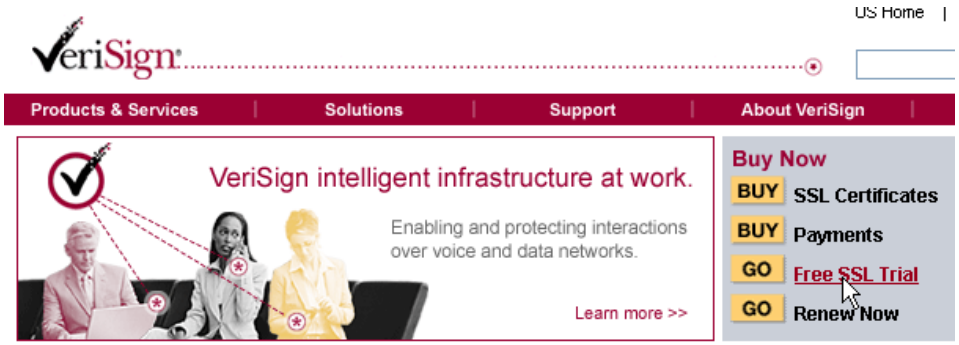
The certification request will be stored in the file “request.pem”. The private key goes into the “private.pem” file. This file is not password protected so take care to back it up securely and delete it from the TOPCALL\Shared directory after the installation has completed. The private key will be stored in an encrypted form inside the TWS configuration.

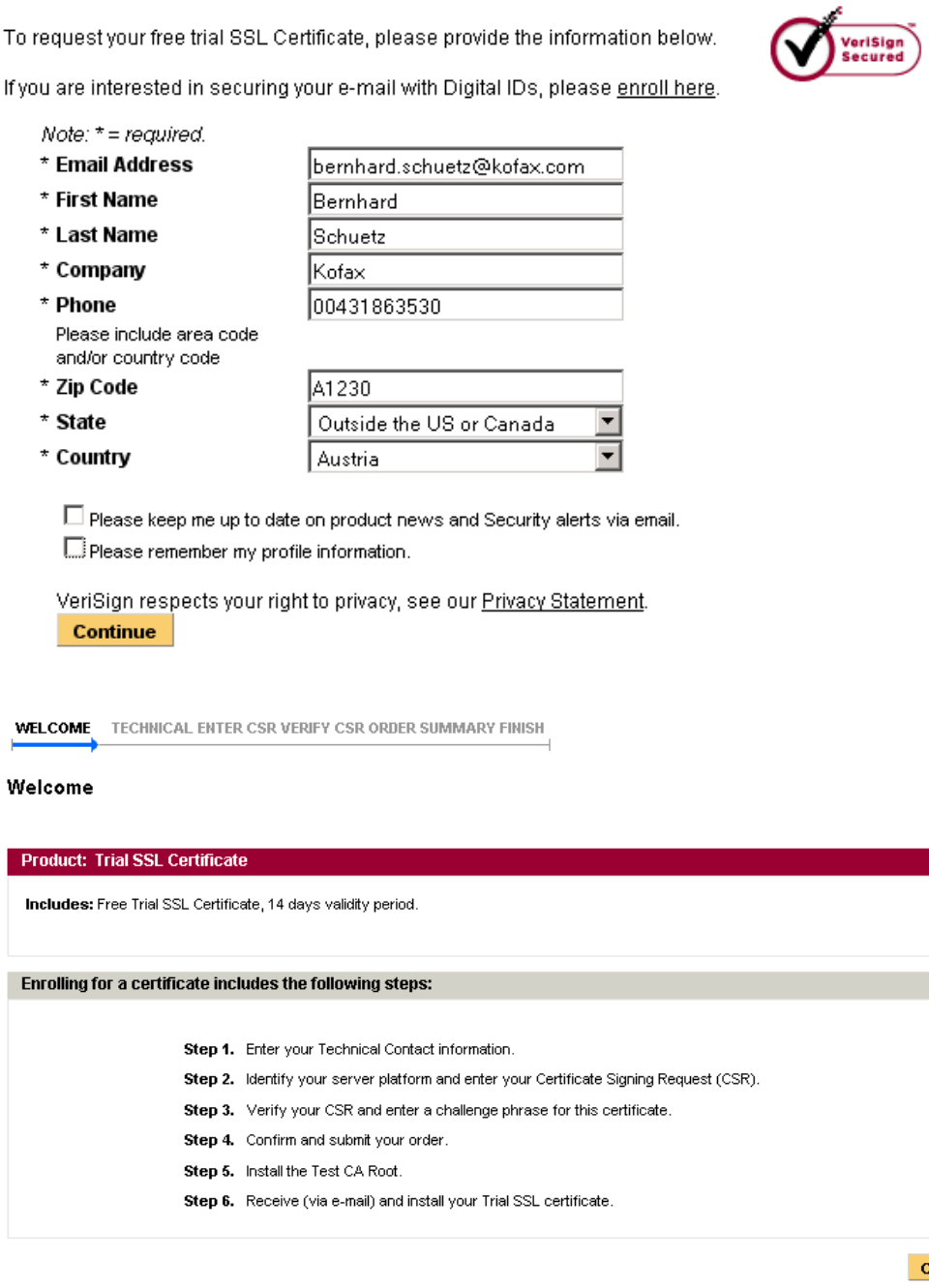
You can now proceed as described in *Step 2: Request a Certificate from a Certification Authority*. Alternatively, one can use the openSSL tool to convert the certificate request into a self-signed certificate for testing. See chapter *Using a Self-Signed Certificate for Testing HTTPS* for further details.

## Step 2: Request a Certificate from a Certification Authority

Open a web browser and navigate to any certification authority, e.g. http://www.verisign.com.

For testing purposes, you can request a test certificate by selecting “Free SSL Trial”





## Free SSL Trial Certificate

To request your free trial SSL Certificate, please provide the information below.

If you are interested in securing your e-mail with Digital IDs, please enroll here.

*Note: * = required.*

| | |
|---|---|
| * **Email Address** | bernhard.schuetz@kofax.com |
| * **First Name** | Bernhard |
| * **Last Name** | Schuetz |
| * **Company** | Kofax |
| * **Phone** (Please include area code and/or country code) | 00431863530 |
| * **Zip Code** | A1230 |
| * **State** | Outside the US or Canada |
| * **Country** | Austria |

☐ Please keep me up to date on product news and Security alerts via email.

☐ Please remember my profile information.

VeriSign respects your right to privacy, see our Privacy Statement.

**Continue**

WELCOME TECHNICAL ENTER CSR VERIFY CSR ORDER SUMMARY FINISH

**Welcome**

**Help**

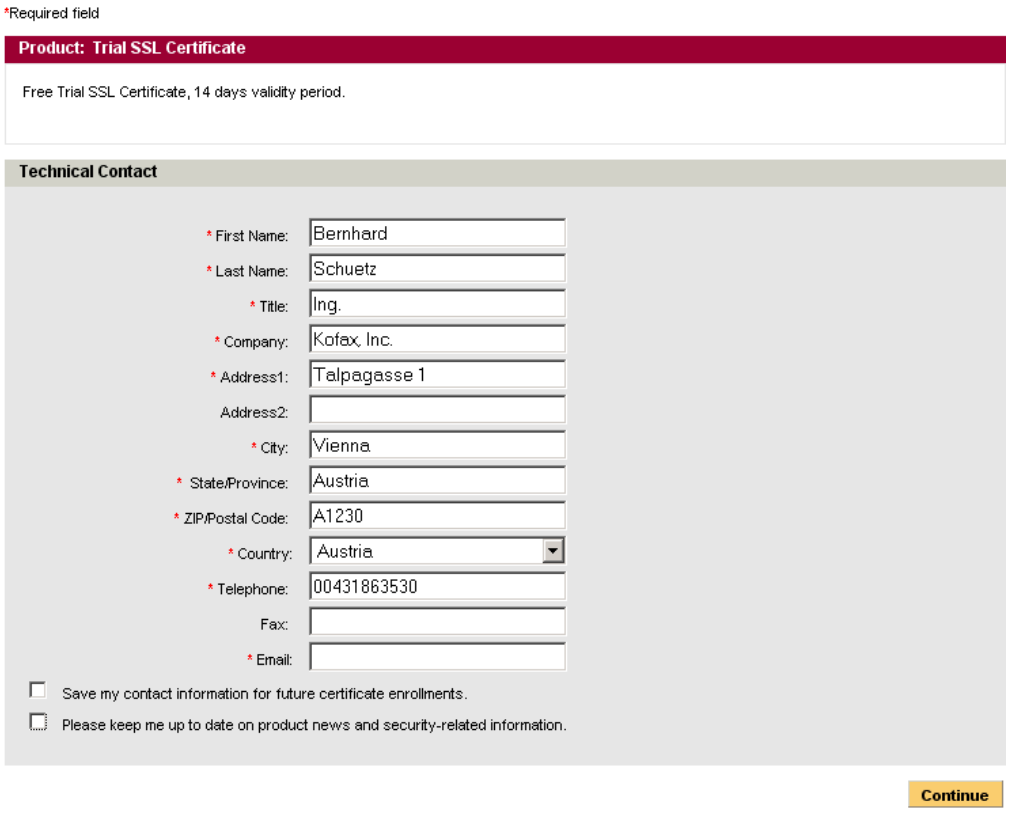
**Product: Trial SSL Certificate**

**Includes:** Free Trial SSL Certificate, 14 days validity period.

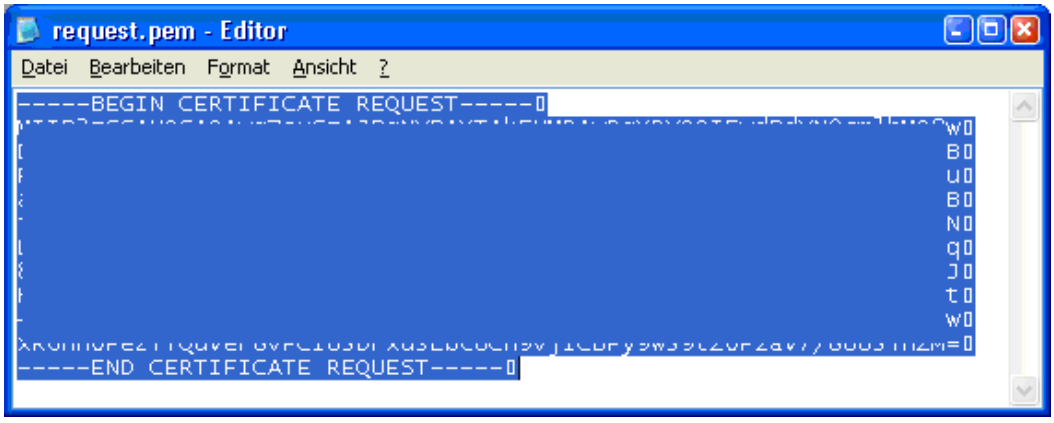
**Enrolling for a certificate includes the following steps:**

**Step 1.** Enter your Technical Contact information.
**Step 2.** Identify your server platform and enter your Certificate Signing Request (CSR).
**Step 3.** Verify your CSR and enter a challenge phrase for this certificate.
**Step 4.** Confirm and submit your order.
**Step 5.** Install the Test CA Root.
**Step 6.** Receive (via e-mail) and install your Trial SSL certificate.

**Continue**

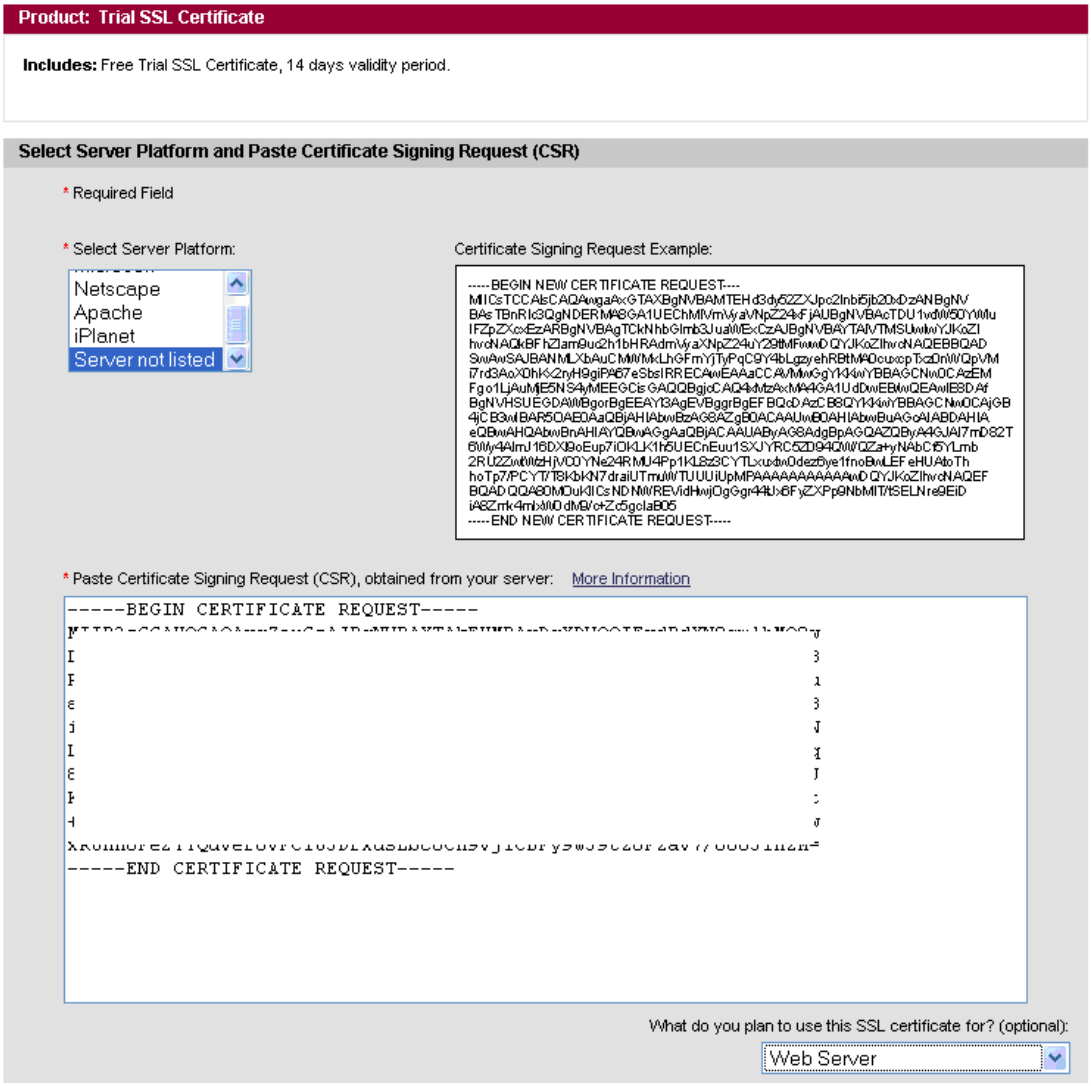



Now open the request.pem with Notepad and select the whole text with Ctrl-A. In the print screen below the request data itself are removed.

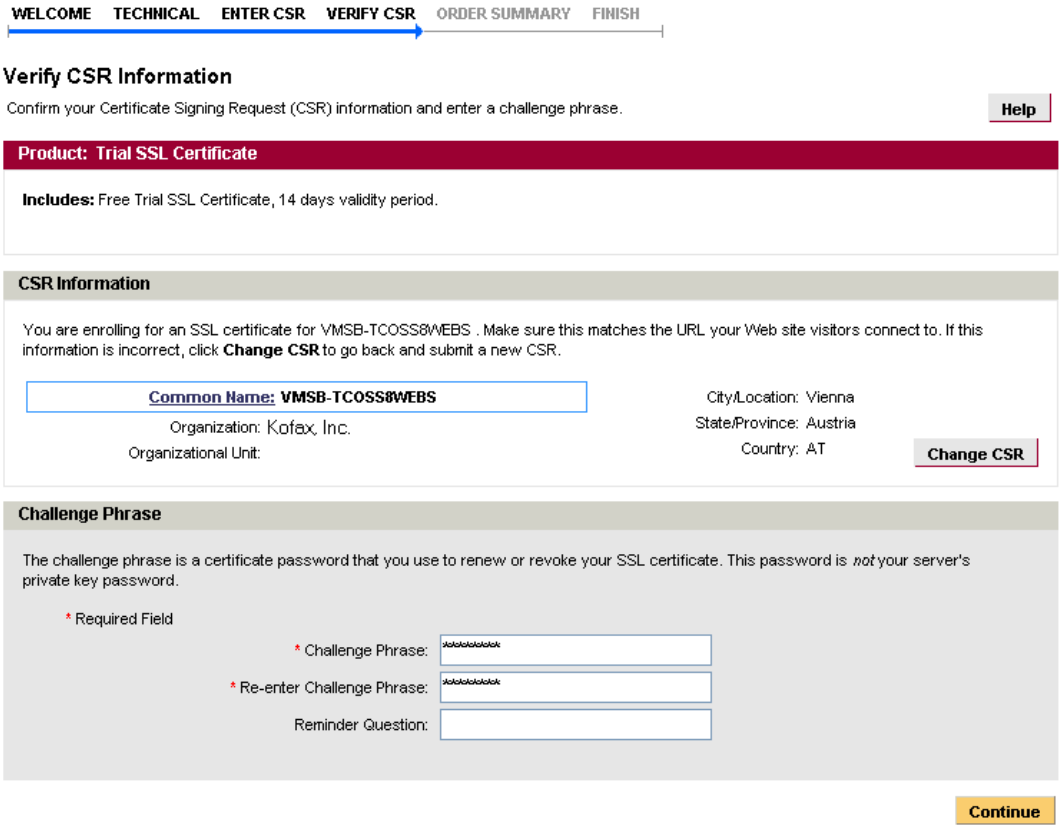


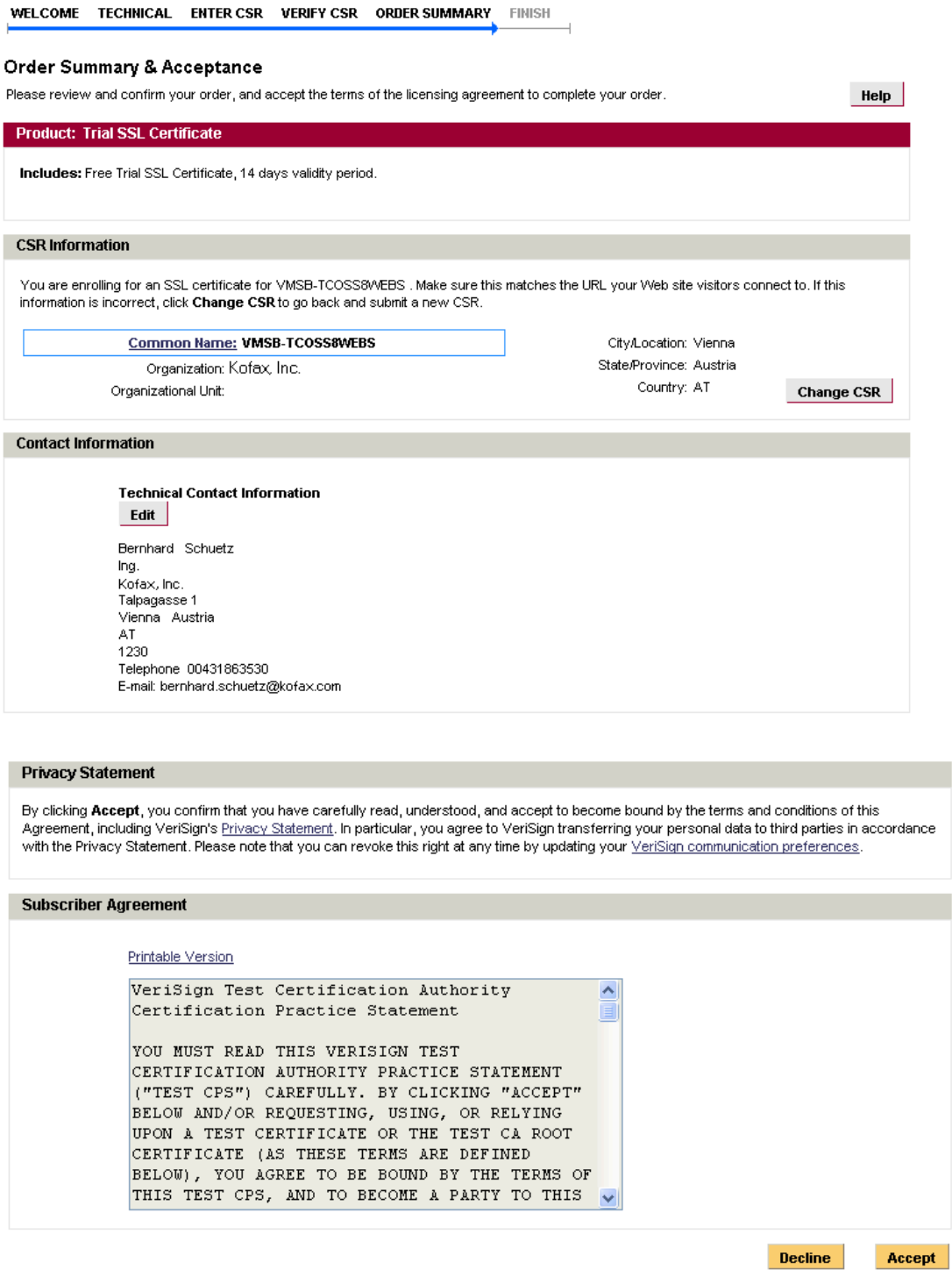

Copy the selected text to the clipboard using Ctrl-C

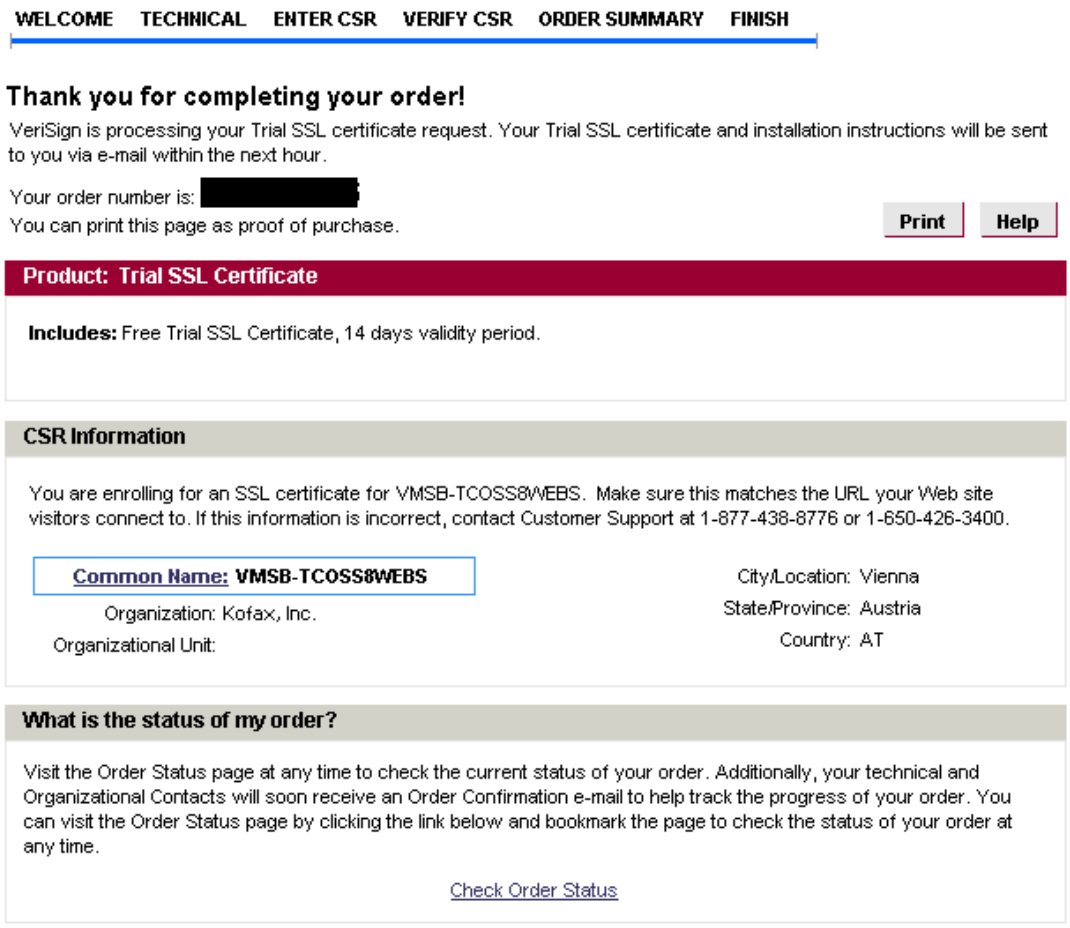
Paste the certification request into the next field of the Certification request wizard




Verify the certification request and enter a challenge password.


WELCOME TECHNICAL ENTER CSR VERIFY CSR ORDER SUMMARY FINISH
Verify CSR Information
Confirm your Certificate Signing Request (CSR) information and enter a challenge phrase.
Help
Product: Trial SSL Certificate
Includes: Free Trial SSL Certificate, 14 days validity period.
CSR Information
You are enrolling for an SSL certificate for VMSB-TCOSS8WEBS . Make sure this matches the URL your Web site visitors connect to. If this information is incorrect, click Change CSR to go back and submit a new CSR.
Common Name: VMSB-TCOSS8WEBS
Organization: Kofax, Inc.
Organizational Unit:
City/Location: Vienna
State/Province: Austria
Country: AT
Change CSR
Challenge Phrase
The challenge phrase is a certificate password that you use to renew or revoke your SSL certificate. This password is not your server's private key password.
* Required Field
* Challenge Phrase:
* Re-enter Challenge Phrase:
Reminder Question:
Continue

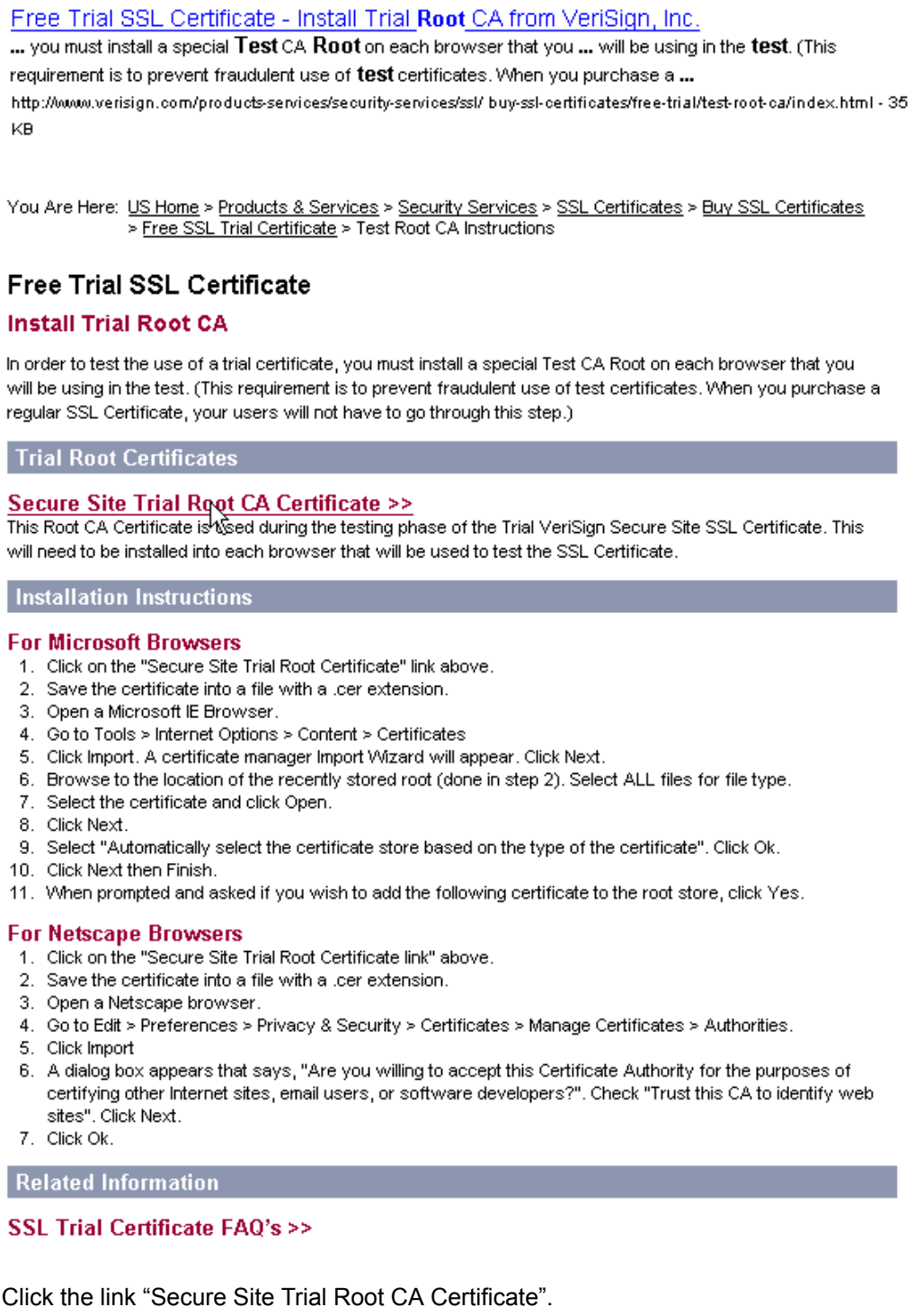

WELCOME TECHNICAL ENTER CSR VERIFY CSR ORDER SUMMARY FINISH
Order Summary & Acceptance
Please review and confirm your order, and accept the terms of the licensing agreement to complete your order.
Help
Product: Trial SSL Certificate
Includes: Free Trial SSL Certificate, 14 days validity period.
CSR Information
You are enrolling for an SSL certificate for VMSB-TCOSS8WEBS . Make sure this matches the URL your Web site visitors connect to. If this information is incorrect, click Change CSR to go back and submit a new CSR.
Common Name: VMSB-TCOSS8WEBS
Organization: Kofax, Inc.
Organizational Unit:
City/Location: Vienna
State/Province: Austria
Country: AT
Change CSR
Contact Information
Technical Contact Information
Edit
Bernhard Schuetz
Ing.
Kofax, Inc.
Talpagasse 1
Vienna Austria
AT
1230
Telephone 00431863530
E-mail: bernhard.schuetz@kofax.com
Privacy Statement
By clicking Accept, you confirm that you have carefully read, understood, and accept to become bound by the terms and conditions of this Agreement, including VeriSign's Privacy Statement. In particular, you agree to VeriSign transferring your personal data to third parties in accordance with the Privacy Statement. Please note that you can revoke this right at any time by updating your VeriSign communication preferences.
Subscriber Agreement
Printable Version
VeriSign Test Certification Authority
Certification Practice Statement
YOU MUST READ THIS VERISIGN TEST
CERTIFICATION AUTHORITY PRACTICE STATEMENT
("TEST CPS") CAREFULLY. BY CLICKING "ACCEPT"
BELOW AND/OR REQUESTING, USING, OR RELYING
UPON A TEST CERTIFICATE OR THE TEST CA ROOT
CERTIFICATE (AS THESE TERMS ARE DEFINED
BELOW), YOU AGREE TO BE BOUND BY THE TERMS OF
THIS TEST CPS, AND TO BECOME A PARTY TO THIS
Decline
Accept




You will get a mail from support@verisign.com including the certificate. Select the parts including -----BEGIN CERTIFICATE----- and -----END CERTIFICATE----- and copy the selected part to the clipboard. Open a new file with Notepad, paste the selected text and save the file with the filename “certificate.pem”.

## Step 3: Getting the Root Certificate

Now you have to get the root certificate for your test certificate. Open again your web browser and navigate to http://www.verisign.com and enter “Test root certificate” within the Search Window:



Free Trial SSL Certificate - Install Trial **Root** CA from VeriSign, Inc.
... you must install a special **Test** CA **Root** on each browser that you ... will be using in the **test**. (This requirement is to prevent fraudulent use of **test** certificates. When you purchase a ...
http://www.verisign.com/products-services/security-services/ssl/ buy-ssl-certificates/free-trial/test-root-ca/index.html - 35 KB

You Are Here: US Home > Products & Services > Security Services > SSL Certificates > Buy SSL Certificates > Free SSL Trial Certificate > Test Root CA Instructions

## Free Trial SSL Certificate

### Install Trial Root CA

In order to test the use of a trial certificate, you must install a special Test CA Root on each browser that you will be using in the test. (This requirement is to prevent fraudulent use of test certificates. When you purchase a regular SSL Certificate, your users will not have to go through this step.)

**Trial Root Certificates**

**Secure Site Trial Root CA Certificate >>**
This Root CA Certificate is used during the testing phase of the Trial VeriSign Secure Site SSL Certificate. This will need to be installed into each browser that will be used to test the SSL Certificate.

**Installation Instructions**

**For Microsoft Browsers**

1. Click on the "Secure Site Trial Root Certificate" link above.
2. Save the certificate into a file with a .cer extension.
3. Open a Microsoft IE Browser.
4. Go to Tools > Internet Options > Content > Certificates
5. Click Import. A certificate manager Import Wizard will appear. Click Next.
6. Browse to the location of the recently stored root (done in step 2). Select ALL files for file type.
7. Select the certificate and click Open.
8. Click Next.
9. Select "Automatically select the certificate store based on the type of the certificate". Click Ok.
10. Click Next then Finish.
11. When prompted and asked if you wish to add the following certificate to the root store, click Yes.

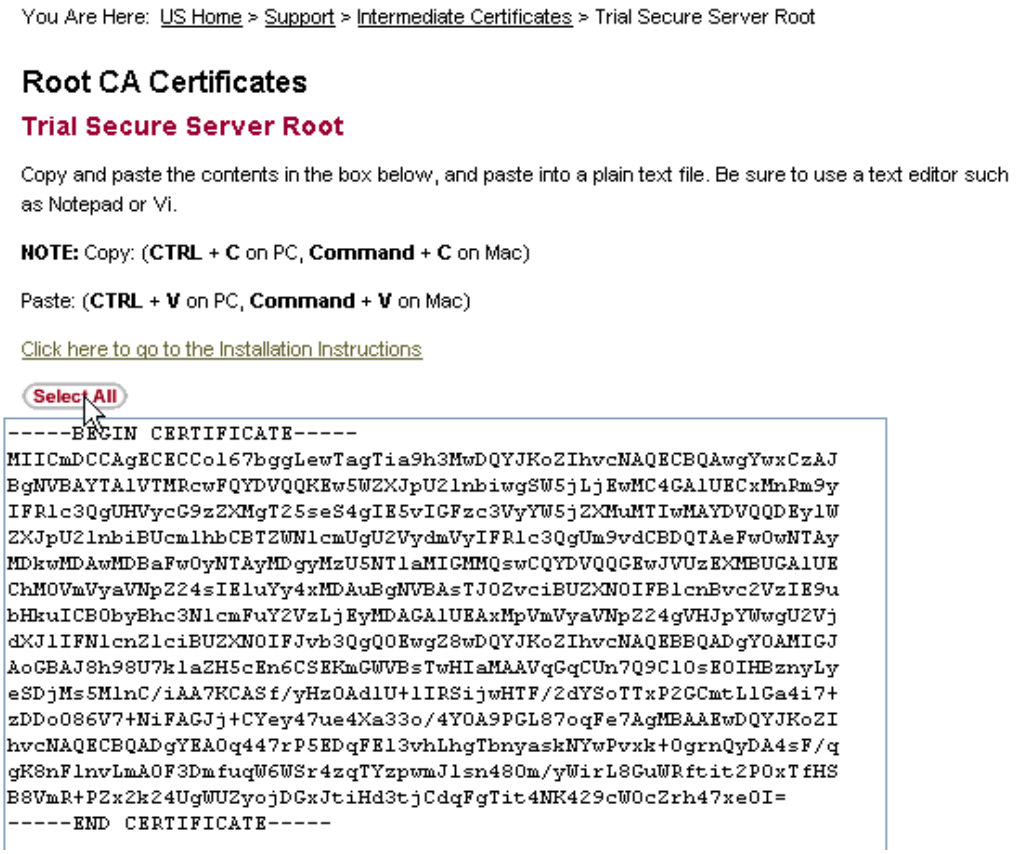
**For Netscape Browsers**

1. Click on the "Secure Site Trial Root Certificate link" above.
2. Save the certificate into a file with a .cer extension.
3. Open a Netscape browser.
4. Go to Edit > Preferences > Privacy & Security > Certificates > Manage Certificates > Authorities.
5. Click Import
6. A dialog box appears that says, "Are you willing to accept this Certificate Authority for the purposes of certifying other Internet sites, email users, or software developers?". Check "Trust this CA to identify web sites". Click Next.
7. Click Ok.

**Related Information**

**SSL Trial Certificate FAQ's >>**

Click the link “Secure Site Trial Root CA Certificate”.




Click “Select All”, copy the whole text to the clipboard using Ctrl-C. Then open again Notepad, paste the contents into a new file and save the file with the filename “rootcert.pem”.

## Step 4: Install the Private Key and the Certificate for the TWS

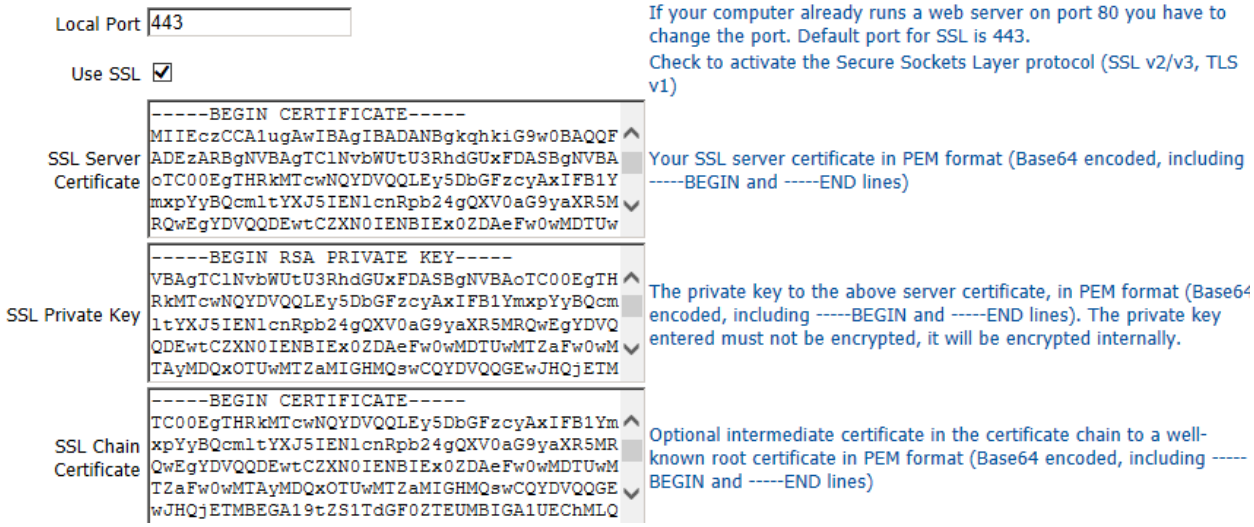
Start the “Configure TCOSS Web Services” from the “Kofax Communication Server”.


In TWS configuration window, go the HTTP Settings tab.

Open the Certificate.pem with Notepad and copy the lines including -----BEGIN CERTIFICATE----- and -----END CERTIFICATE----- to the clipboard. Paste it then into the field “SSL Server Certificate”

Open the private.pem with Notepad and copy the lines including -----BEGIN RSA PRIVATE KEY----- and -----END RSA PRIVATE KEY----- to the clipboard. Paste it then into the field “SSL Private Key”

Set Local Port to 443 (HTTPS).




Optionally open the Rootcert.pem with Notepad and copy the lines including -----BEGIN CERTIFICATE----- and -----END CERTIFICATE----- to the clipboard. Paste it then into the field “SslChainCertificate”.

This will allow you to import the root certificate in the Internet Explorer later when being connected to TWS instead of importing it from a file: Select “File” – “Properties” from the IE menu and press the “Certificates” button. Go to the tab “Certification Path” in the “Certificates” window, select the root certificate at the top of the tree view and click “View Certificate” and then “Install Certificate”.

## Step 5: Install the Root Certificate on the SoapClient Machine

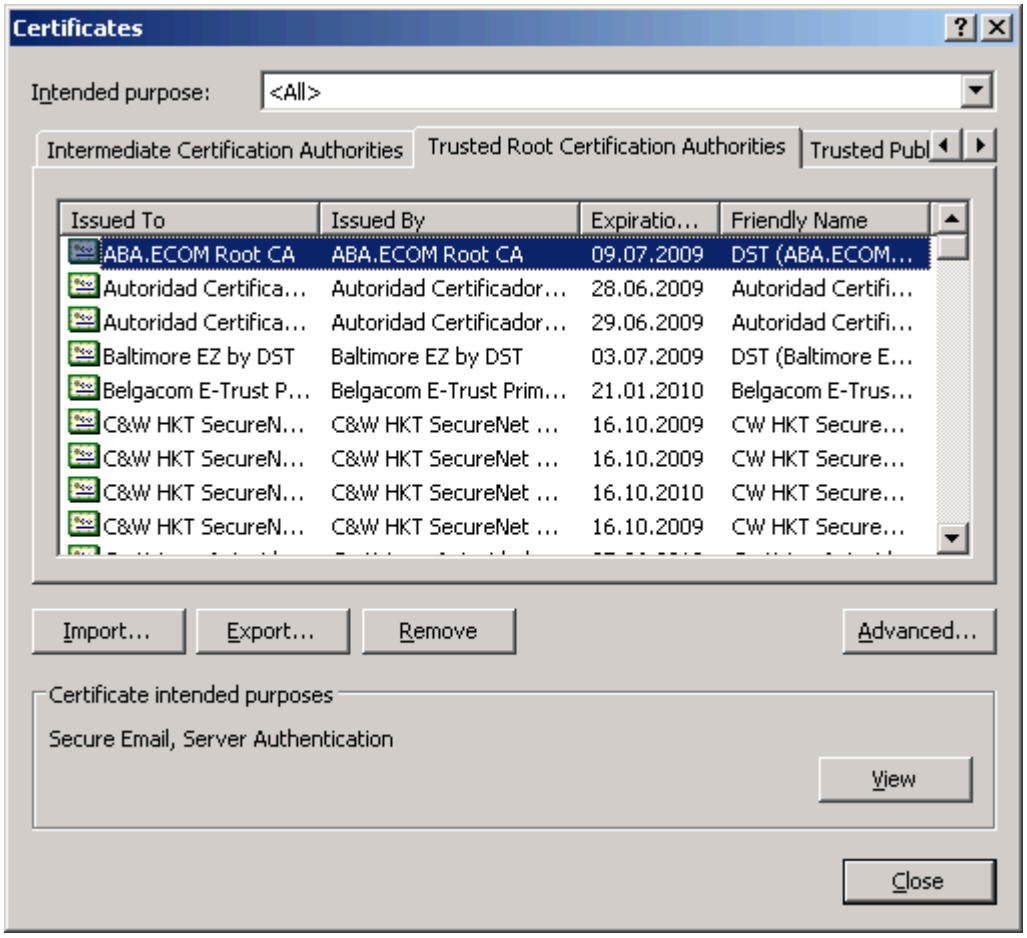
Now you have to install the root certificate on that machine where you run the web service application (the SoapClient calling TWS). Check within your Soapclient’s documentation, how the SoapClient does the lookup for the root certificates and how the root certificates must be installed.

PocketSOAP, for example uses the certification store of Internet Explorer (see installation instructions below) and you must use the https:// prefix within the URL of the SOAP method to use HTTPS transport. Visual Studio 2003 VB.NET also uses the certification store of Internet Explorer.

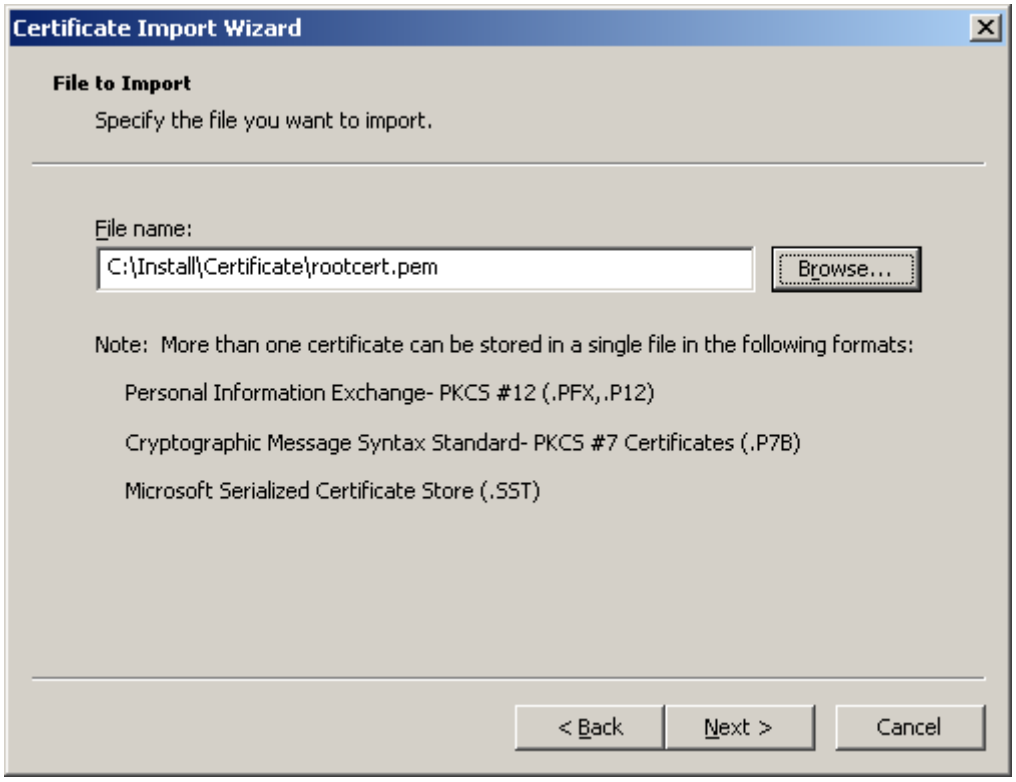
Additionally when using the Simulated Calls from within a web browser, you must also install the root certificates for the used web browser

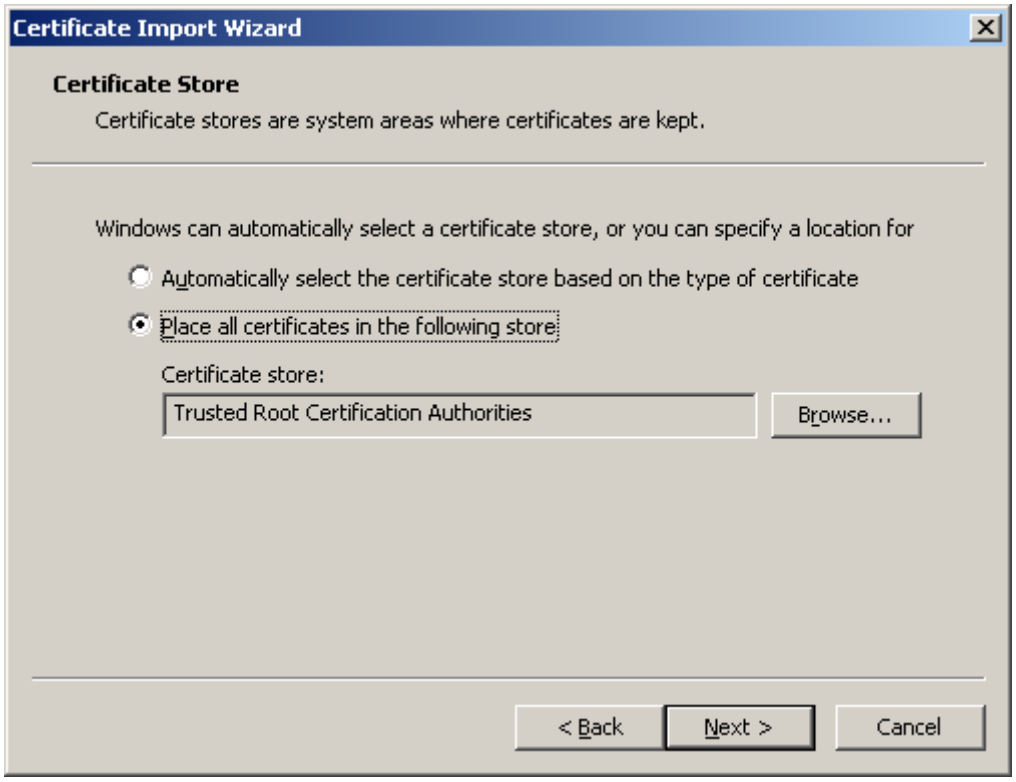
### Step 5.1: Installing the Root Certificate for Internet Explorer:

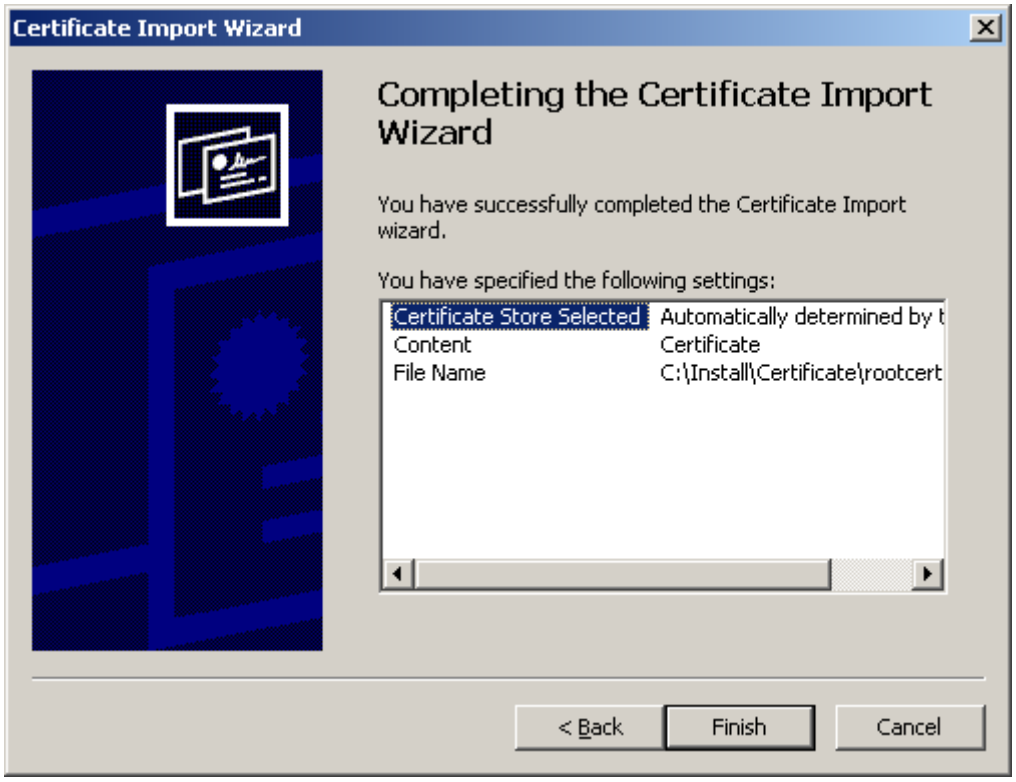
Open Internet Explorer and select Tools – Internet Options – Content – Certificates. Go to the Trusted Root Certification Authorities tab.

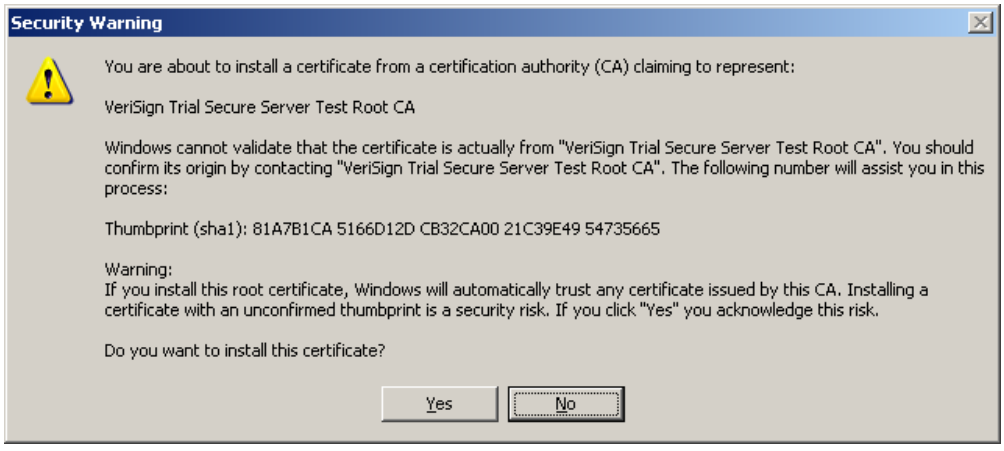



Click Import. The Certificate Import Wizard will start. Click Next. Browse to the rootcert.pem and click Next.

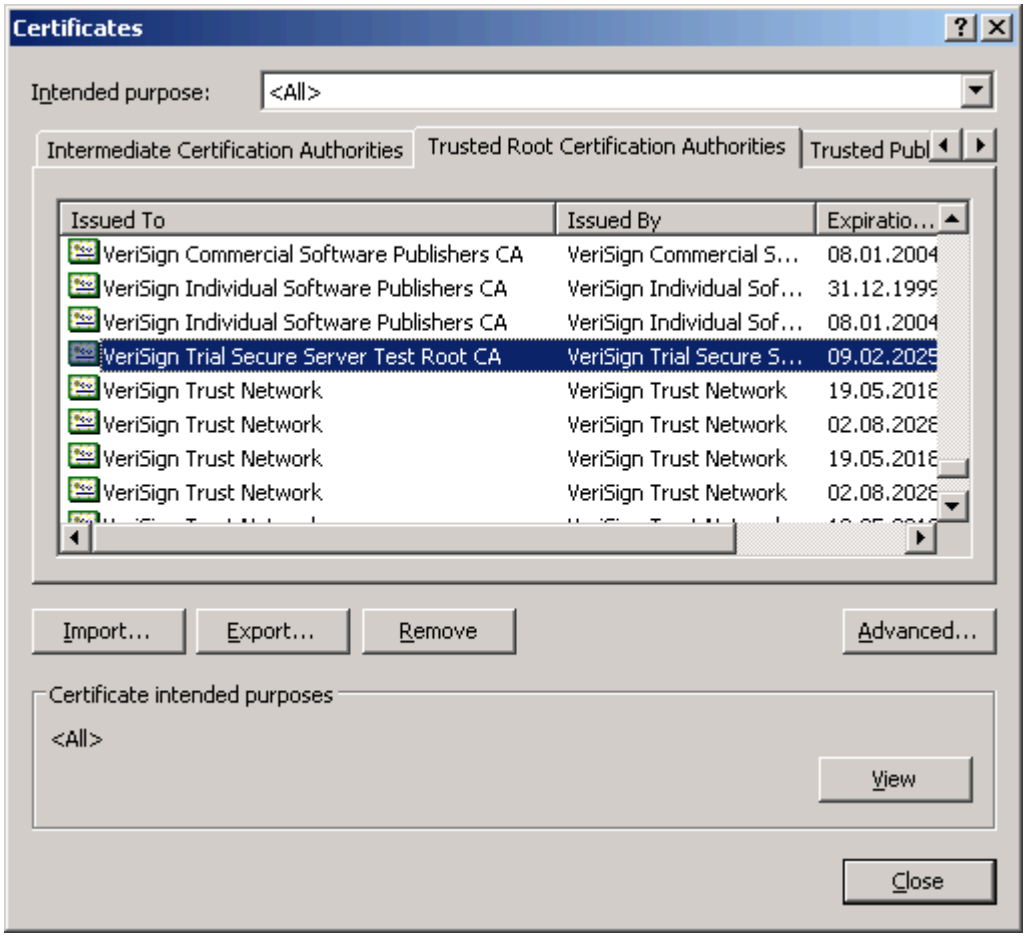

Certificate Import Wizard
File to Import
Specify the file you want to import.
File name:
C:\Install\Certificate\rootcert.pem
Browse...
Note: More than one certificate can be stored in a single file in the following formats:
Personal Information Exchange- PKCS #12 (.PFX,.P12)
Cryptographic Message Syntax Standard- PKCS #7 Certificates (.P7B)
Microsoft Serialized Certificate Store (.SST)
< Back
Next >
Cancel


Certificate Import Wizard
Certificate Store
Certificate stores are system areas where certificates are kept.
Windows can automatically select a certificate store, or you can specify a location for
Automatically select the certificate store based on the type of certificate
Place all certificates in the following store
Certificate store:
Trusted Root Certification Authorities
Browse...
< Back
Next >
Cancel

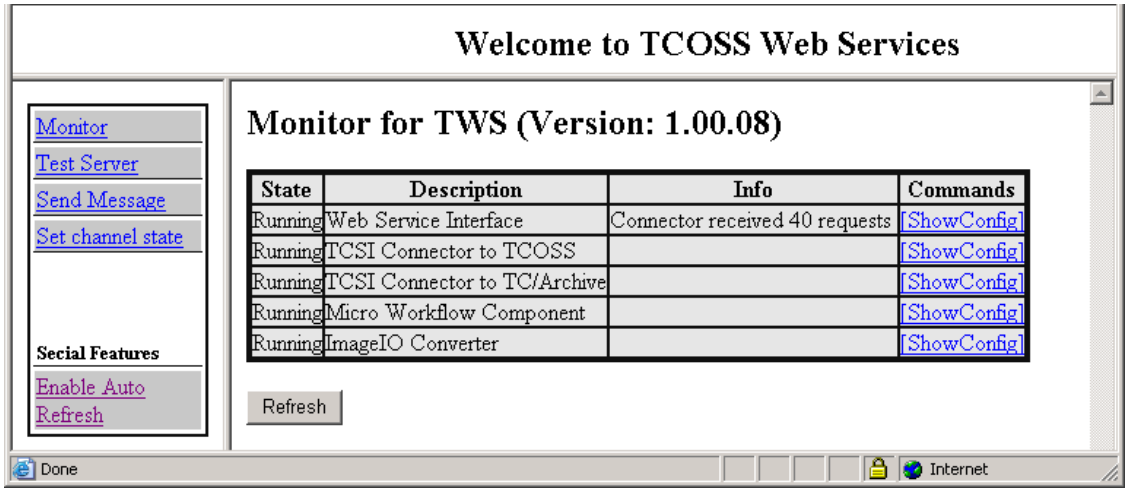



Click Yes. Verify that the root certificate is installed and close the window

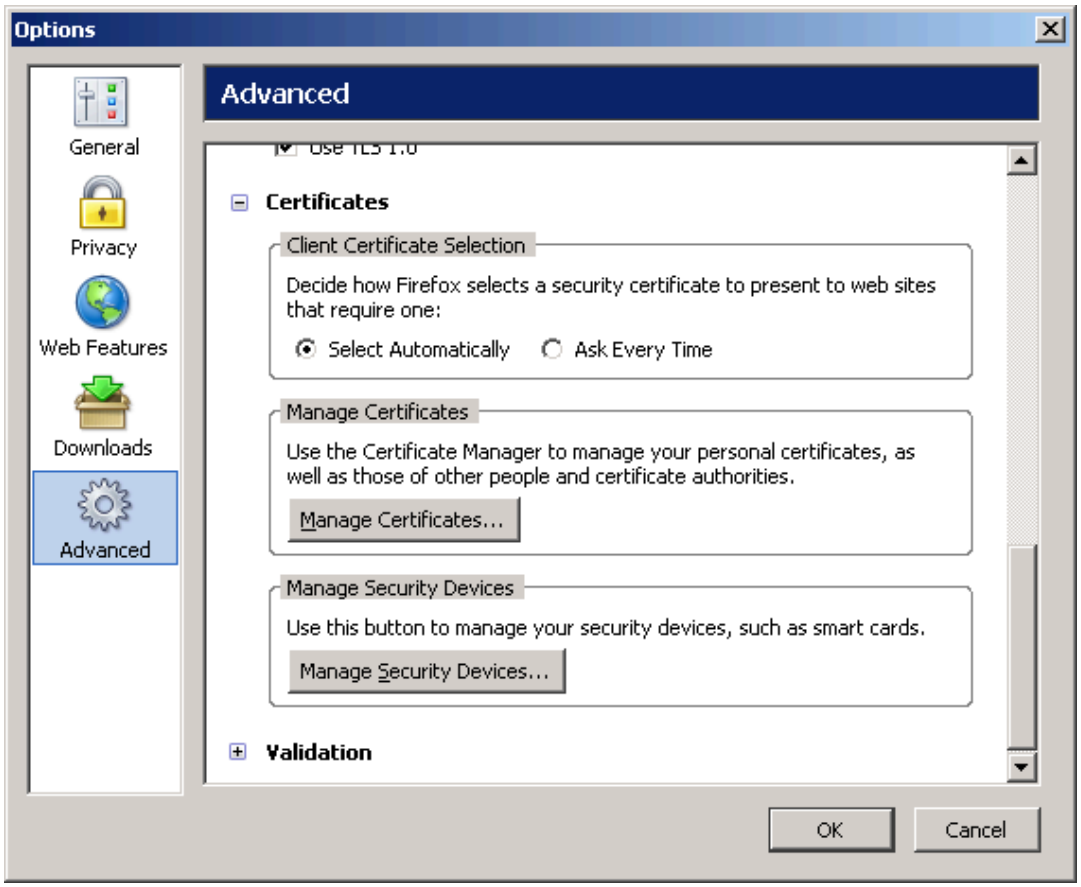



Verify the successful installation by opening the Internet Explorer web browser and navigate to https://<tcosswebservicemachine>:<port>/file/index.html

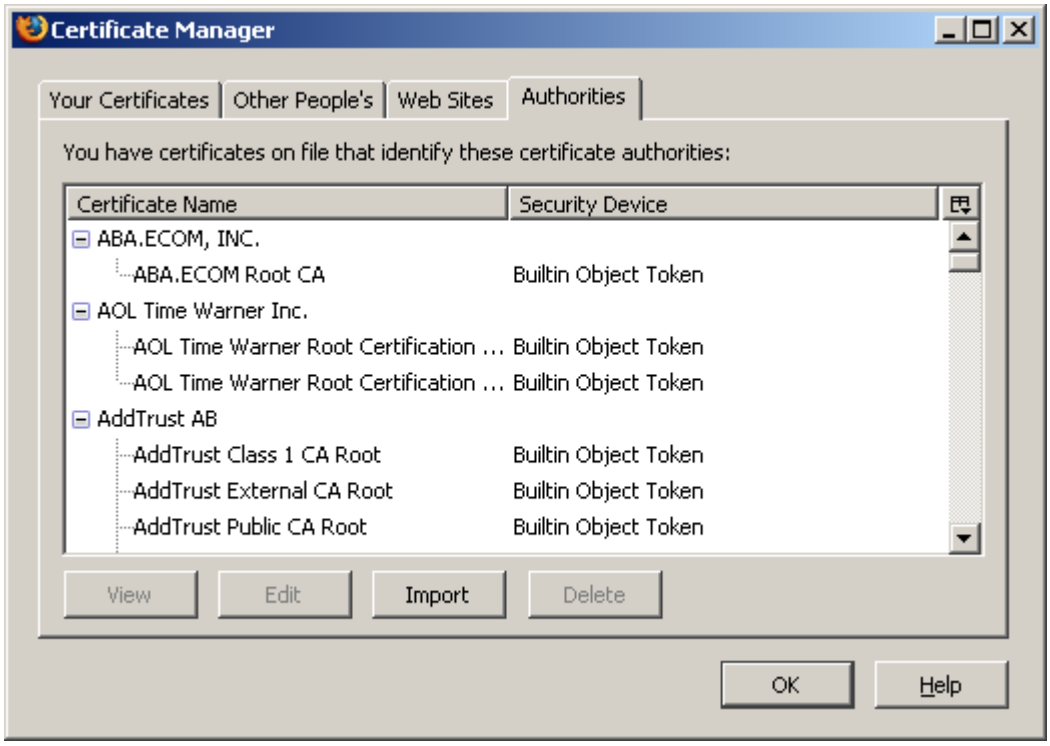


## Step 5.2: Installing the Root Certificate for Mozilla Firefox:

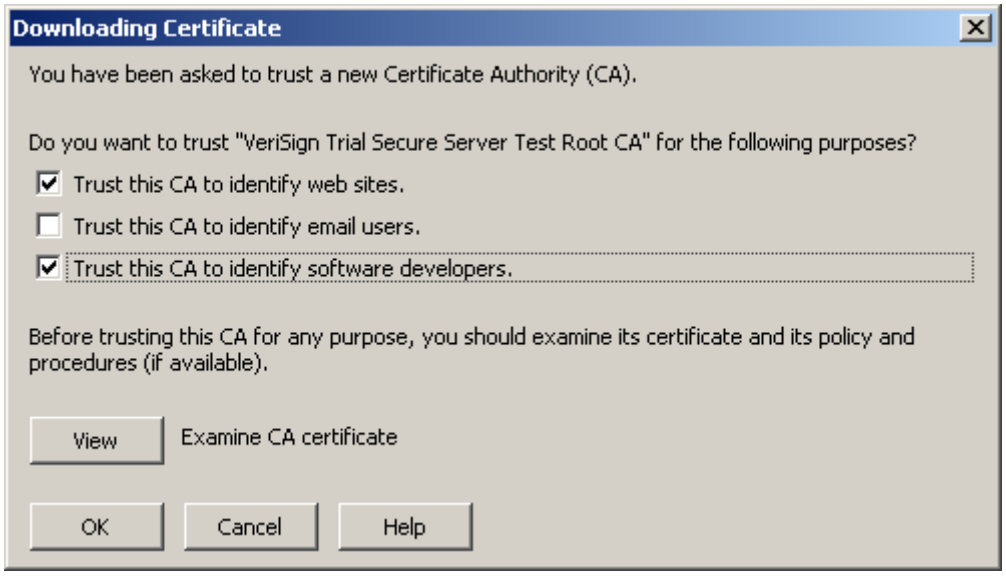
Select Tools – Options – Advanced. Expand Certificates and click Manage Certificates



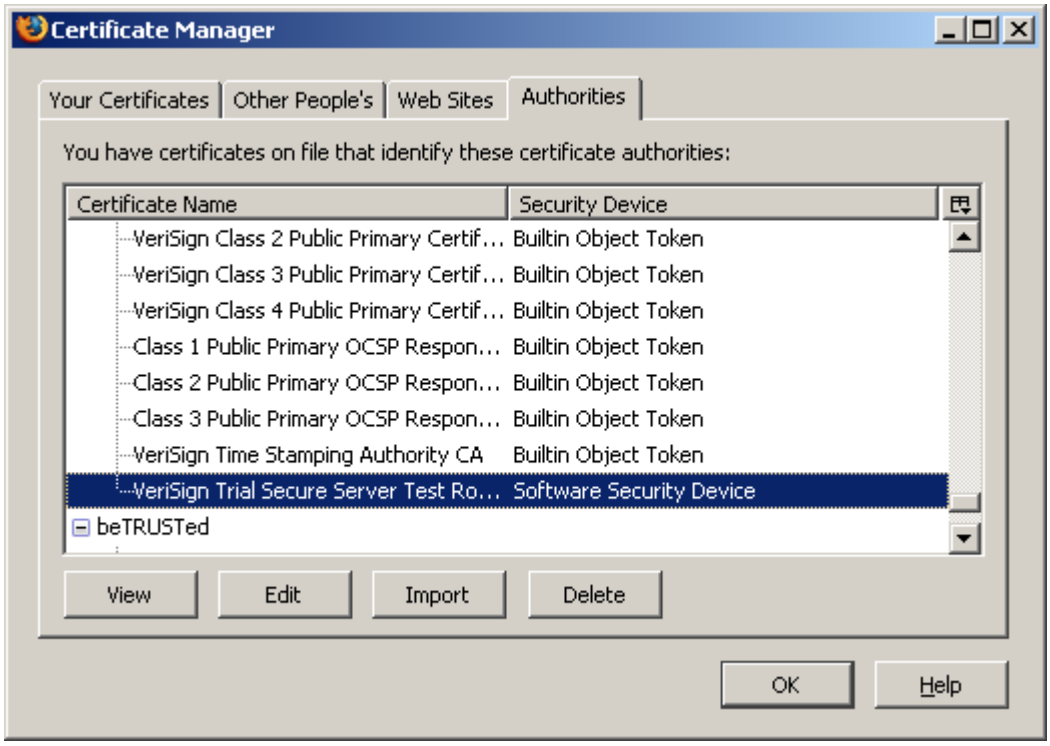

Select the Authorities tab and click Import




Reselect the rootcert.pem file



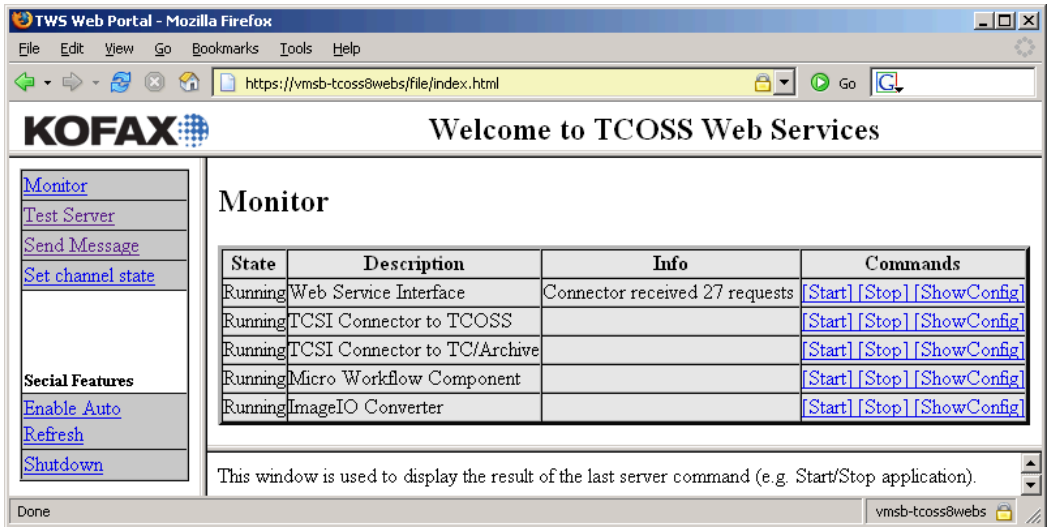

Select the 2 checkboxes listed above and click OK




Click OK to close all windows.

Verify the successful installation by opening the Firefox web browser and navigate to

https://tcosswebservicemachine/file/index.html



## Using a Self-Signed Certificate for Testing HTTPS

Instead of requesting a certificate from a certification authority you can also use the openSSL tool to get a "self-signed" certificate based on your certification request. Start with "*Step 1: Create a Private Key and a Certification Request*". Then execute the following commands as an administrator:

```
C:\topcall\SHARED>set OPENSSL_CONF=openssl.cnf
openssl x509 -req -in request.pem -signkey private.pem -out certificate.pem -days 3650
Loading 'screen' into random state - done
Signature ok
subject=/C=AT/ST=Austria/L=Vienna/O=Kofax/CN=VMSB-TCOSS8WEBS
Getting Private key
```

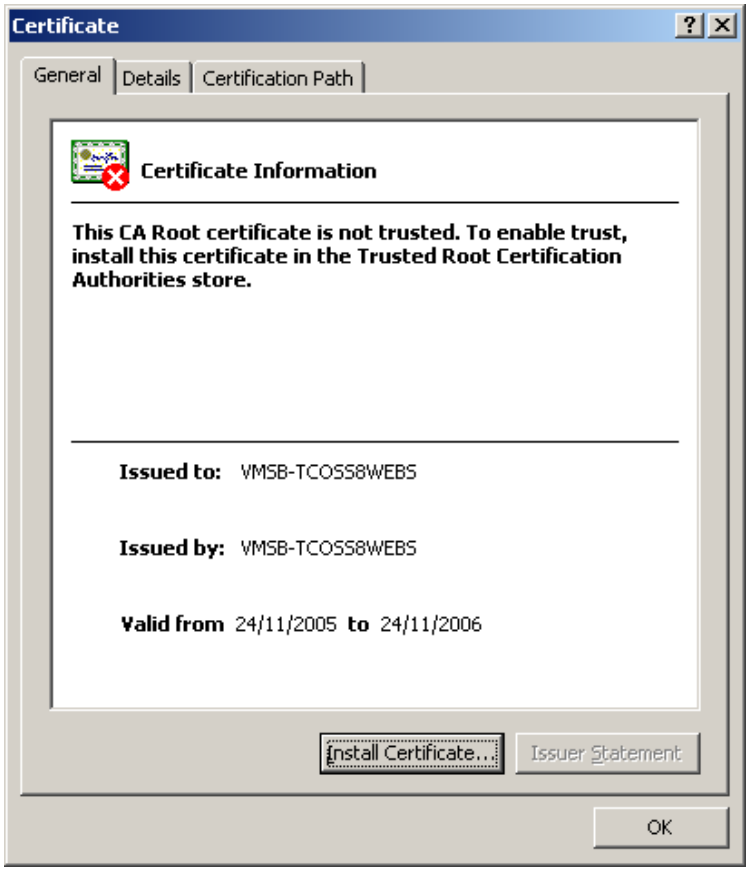
The certificate will be stored in the file "certificate.pem". You can view its details with the command:

```
openssl x509 -in certificate.pem -noout -text
```

For a complete documentation of openSSL options see http://www.openssl.org/docs/apps/openssl.html.

Now do "Step 4: Install the Private Key and the Certificate for the TWS". Start TWS and connect from your client machine (where you install the SoapClient application) to the TWS Server with Internet Explorer (https://YourHost/file/index.html). You will see an alert, proceed with "Yes":


To install the test certificate as CA root certificate and to avoid the alert, select "File" – "Properties" from the IE menu and click the "Certificates" button (while still being connected to TWS):




Click Install Certificate to start the certificate import wizard where you simply proceed with “Next” until a final warning, then choose “Yes” to install your test certificate:

# Blacklist / white-list support

TWS supports an IP packet filter that contains a list of rules. Each rule starts with an allow/deny keyword followed by an IP-address range that is compared with the source IP-address of the request.

If no rule is defined, then all requests are allowed. If there are rules defined but no rule matches, the request is denied. Else, the first matching rule defines whether the request is allowed or denied.

IP-ranges use the Classless Inter-Domain Routing (CIDR) notation that defines the number of network bits as integer value instead of using a network mask. In particular, IPv4 ranges are represented as described in RFC 4632 and IPv6 addresses are represented as described in chapter 2.3 of RFC 4291.

The IP filter syntax according augmented BNF syntax (RFC 5234) is shown below:

```
IP-filter = *filter-line
filter-line = ("Allow" / "Deny") 1*" " ip-address ["/" network-bits] CRLF
ip-address = ipv4-address / ipv6-address
```

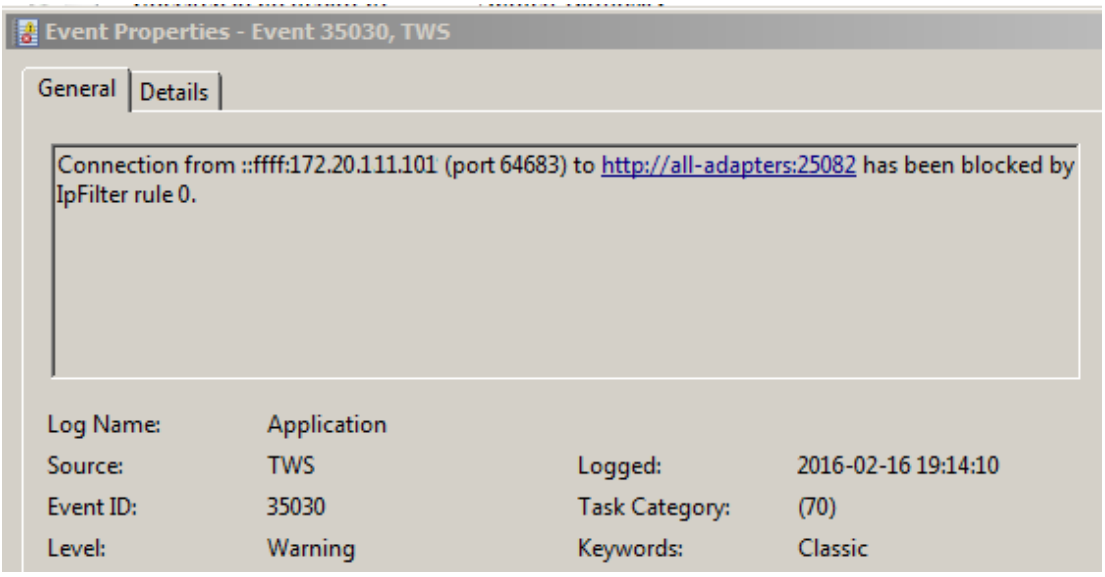
Here is an **Example**:

```
Deny 10.20.0.100
Allow 10.20.0.0/24
Allow 1:2:3::/80
Allow ::1
```

This configuration allows the following source addresses:

- IPv4 ranges: 10.20.0.0 – 10.20.0.99, 10.20.0.101 – 10.20.0.255.
- IPv6 range: 1:2:3:: - 1:2:3:0:0:FFFF:FFFF:FFFF
- Requests from local host which typically uses the IPv6 loop-back address ::1

If a request is denied, an event log entry with ID 35050 is created and the TCP/IP connection is closed without any response. An example event log entry is shown below.



**Note**

- If further requests from the same IP are blocked within 1 minute after such an event log entry, no additional event log entry will be created.
- IPv4 addresses may be written as IPv4 notation (a.b.c.d) or IPv4 mapped IPv6 address (for example ::ffff.a.b.c.d). Here are some example of equivalent IPv4 addresses:
  - 10.20.30.40 is equivalent to ::FFFF:10.20.30.40 is equivalent to ::FFFF:A14:1E28
  - 1.2.0.0/16 is equivalent to ::FFFF:1.2.0.0/112 is equivalent to ::FFFF:102:0/128

# Installation Details

TWS will be installed in directory C:\topcall\tws\00. This chapter provides a more detailed description about the installation. If you do not want to know these details, you can safely skip this chapter.

Configuration information is stored in the following places:

- Registry (HKLM\Software\TOPCALL\TWS) stores information about the windows user credentials that is used to run TWS.
- The XML file C:\TOPCALL\tws\00\config\SolutionConfig.xml contains all data that have been entered in the TWS configuration page.

The table below provides a brief explanation about the used files and directories within TWS installation directory:

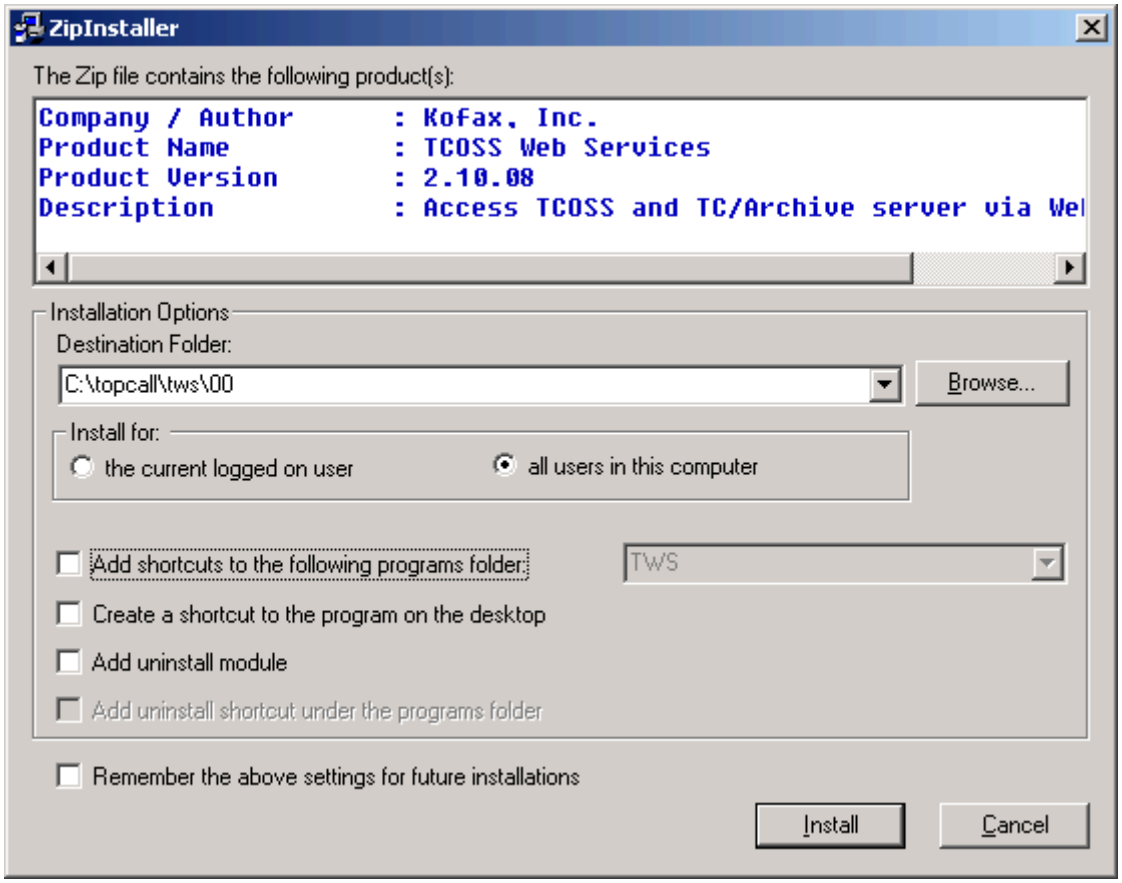
| Files | Description |
|---|---|
| Configure.bat | Can be called to start the configuration tool |
| Run.bat | Can be called to run TWS from command line (without TCSRV) |
| api*.xsd | XML Schema files for data types used in Web Services |
| api*.xml | Source definition of Web Service functions. This file lists all functions and input/output parameters for each call. Details about the parameters are defined in the schema files. |
| api\wsdl.xslt | XSLT transformation generating WSDL files. |
| api\template*.* | Template for additional custom specific web service functions |
| Bin*.* | This directory contains the binaries (and some XML files) of the application |
| Config\SolutionConfig.xml | This file contains all configuration parameters that can be changed by the configuration tool. |
| Config\InternalConfig.xml | Internal configuration file that will be executed by the binaries. This file is created from SolutionConfig.xml by using an XSLT transformation. |
| Config\Custom.xslt | Hold the XSLT or TSL code used for customization features |
| Config\CustomConfig.xsd | XML Schema for optional customer specified configuration data. |
| Trace*.* | This directory contains all generated traces |

| Files | Description |
|---|---|
| Web*.* | These files are presented on the web portal. All files in the Web directory and any sub-directory can be accessed. E.g. a URL like http://localhost:[port]/file/xxx.html opens the file xxx.html in that directory. |
| Web\stylesheets*.xsl | Style sheets for converting the XML response in a simulated web service call into a nice HTML page. |
| xcd\Create_Config.xslt | XSLT transformation file that transforms the easy SolutionConfig.xml file into the internal configuration file InternalConfig.xml. |
| xcd\SolutionConfig.xsd | Schema for SolutionConfig.xml. This schema is used to generate the HTML form in the Configuration Tool. |
| Xcd\SolutionConfigDefault.xml | Template for new configurations. It is used if the file SolutionConfig.xml does not exist. |

**Note** The information above should be handled as a hint only. Do not rely on these structures. It is possible that names and meanings of files will be changed in future releases without further notice.

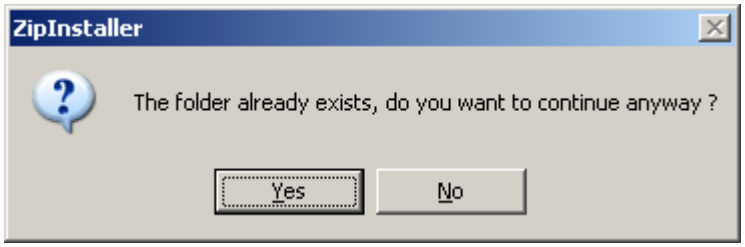
# Module Update

In some cases bug fixes or new features will be distributed in the form of a module update. This is a self-extracting ZIP file “TWS.exe”. Stop any active TWS instances and double-click this file to start the update:

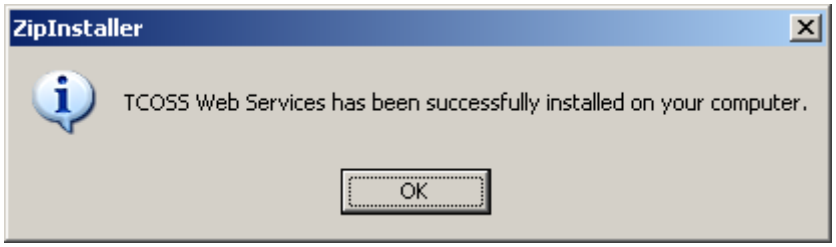



The destination “C:\topcall\tws\00” needs to be changed only if multiple instances of TWS are installed. The first instance goes into subdirectory “00”, the second into “01”, and so on.

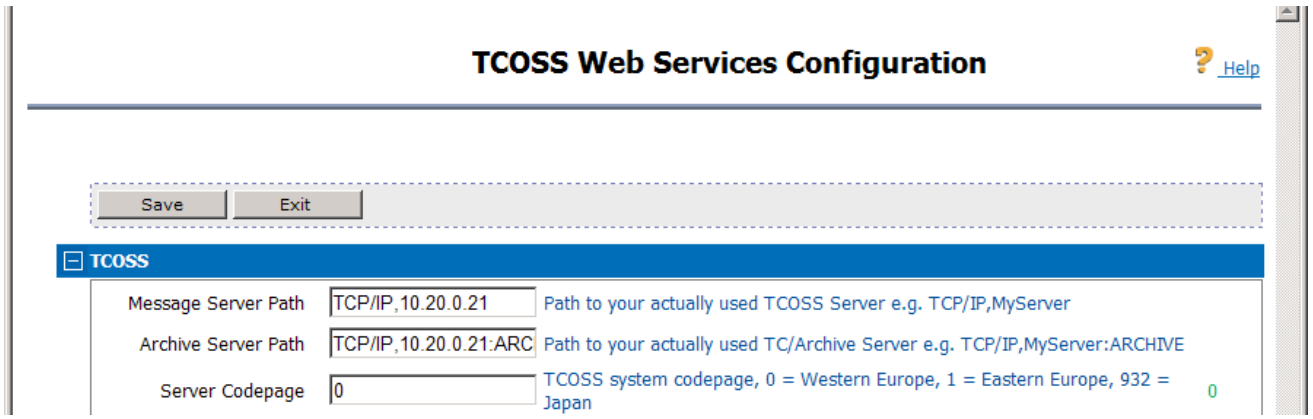
Press the “Install” button, confirm to overwrite an existing folder




and wait one or two seconds until this message is displayed:




After installing the update start the “Configure.bat” file in the installation directory, e.g. “C:\topcall\tws\00”:




Leave the configuration by pressing “Save” or “Exit”. It is necessary to go through the configuration window even if the configuration settings are not changed, because the internal configuration file (InternalConfig.xml) generated from these values by an XSLT transform has to be updated.

# Backing Up Configuration Data

TWS – in difference to other Kofax applications – stores most of its configuration data in XML files (see chapter “Installation Details”). Only for the integration into the TCSRV service some registry entries are used.

Additionally, some data (password, private key in case of using HTTPS) are stored encrypted within the XML configuration files. For encrypting those data also some machine-specific settings are used.

Therefore, it is not possible to re-import those encrypted data into a new machine, just by simply restoring the XML configuration files.

So – after finishing a customer installation please backup following files:

- Registry backup of HKLM\SOFTWARE\TOPCALL
- All certificate specific files (private.pem, certificate.pem, rootcert.pem, request.pem) stored in the C:\TOPCALL\SHARED folder
- The password configured for the KCS user (if configured)
- All files stored in the C:\TOPCALL\TWS\00\Config directory.

- If a WSDL file is used for additional customer specific web service functions (see Custom Web Services) files C:\TOPCALL\TWS\00\api\custom.* should be included.

After restoring the XML config files please rerun the Configure.bat and enter all encrypted data again

(Private key, KCS password).

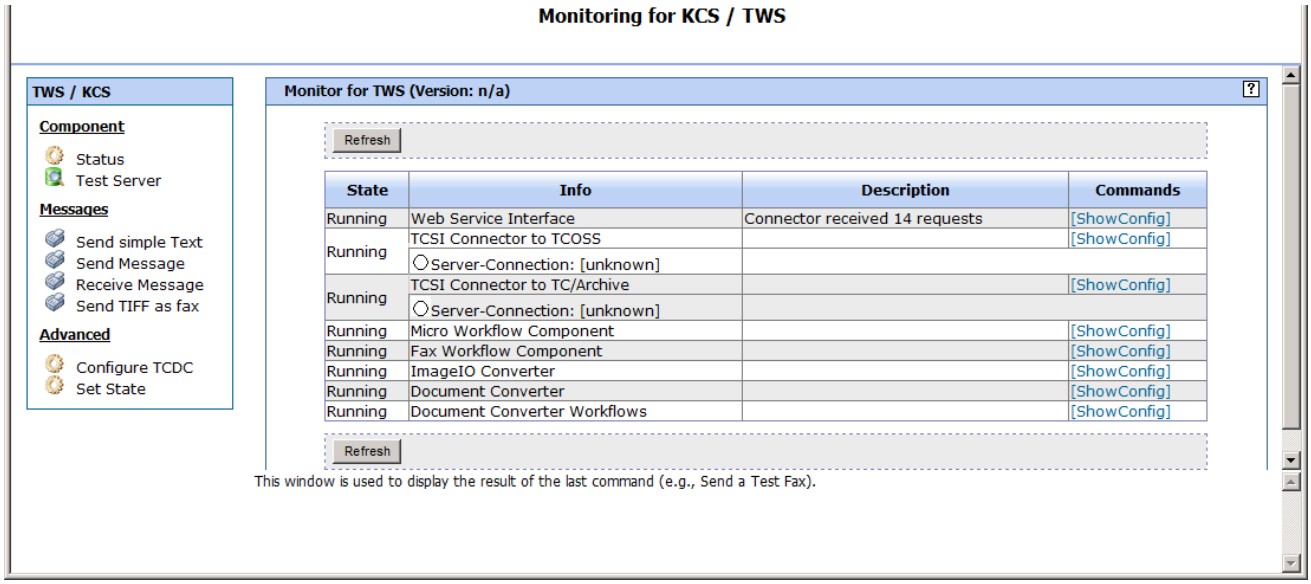
Chapter 4

# Web Portal

The Application provides a web portal which can be accessed via any standard internet browser (tested with Internet Explorer and Mozilla). The start address of this portal is:

http[s]://<Computer>[:Port]/file/index.html

A screenshot of the portal page is shown below:



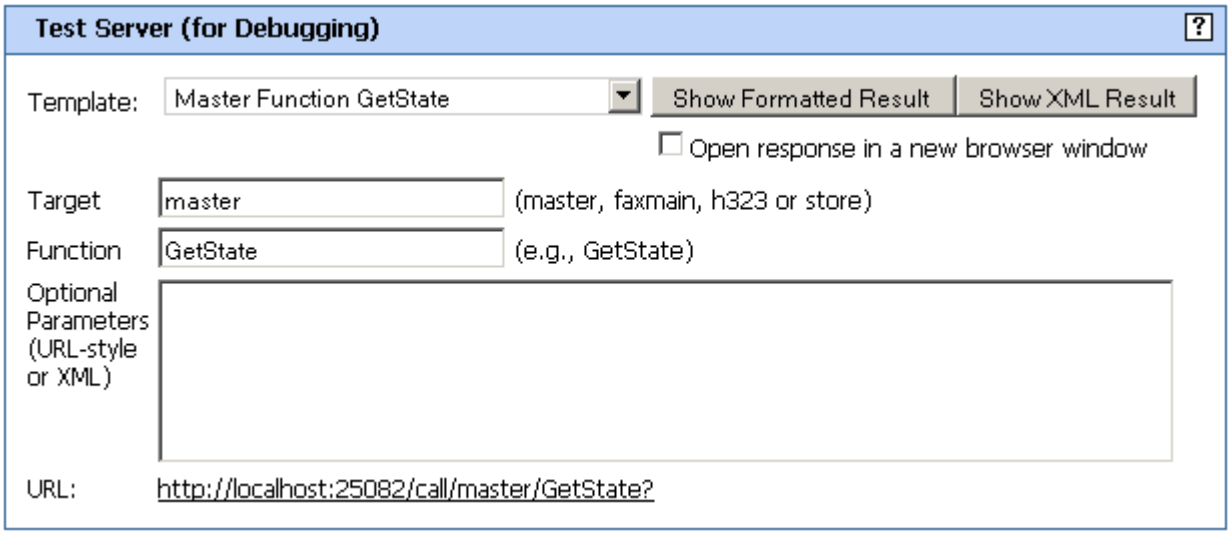

If you get this response, at least the HTTP connector is working. This page is not intended for the end user. It provides the following tools and samples for integrators:

- Status: State overview of the application (see screen shot above)
- Test Server: Simple web service calls can be tested within the web browser.
- Send simple Text: An HTML form to send simple messages using TCOSS server
- Send Message: An HTML form to send any message (including attachments and document conversion) using TCOSS server
- Receive Message: An HTML form to receive a message from a TCOSS server (Note: By default the “DIST” queue is polled. Make sure that you do not have TCfW running with distribution mode active – this could cause the Receive Message not to receive the message for several minutes)
- Send TIFF as fax: A simple HTML form to send simple fax messages using TCOSS server
- Configure TCDC: Allows to configure Document Converter
- Set State: A simple HTML form to start or stop TCOSS channels

Click Refresh to update the status monitor.

# Test Server

If you click on “Test Server” in the left frame, you will get the following page

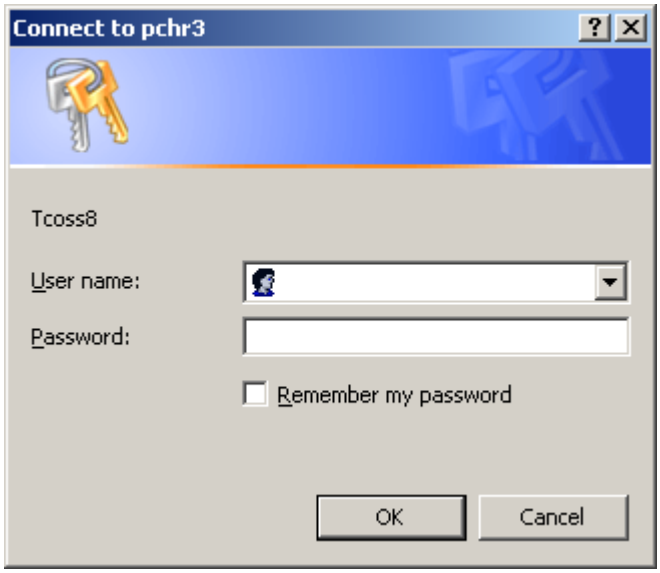



This page provides a list of predefined function calls to TCOSS. These calls can be selected with the dropdown box “Templates”. Whenever you select an entry from this dropdown box, all other fields below will be updated according to the selected template. If the other fields are not updated automatically, check whether your web browser is enabled for JavaScript.

On the right side of “url:” you can see the generated URL that represents the function call (using Simulated Web Service Calls as described in chapter Simulated Web Service Calls). If the check box “Open response in new browser window” is checked, the result will be written into a new browser window. Otherwise, it will be written into the frame on the bottom of the page.

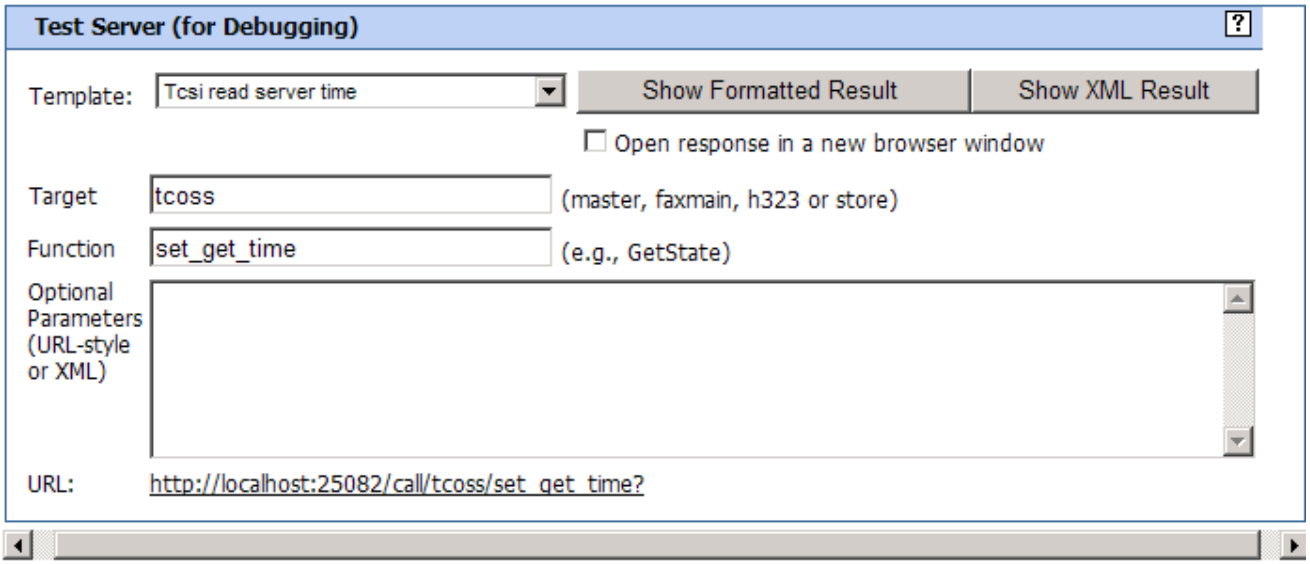
If you Press button “Show formatted result” the result of the function call will be converted using a style-sheet or content type if one of them is available. The button “Show XML result” always returns the response as XML. This button is very suitable to see the complete response for each function.

You can now try to start using the different templates provided by the test client. You can also change any parameters to learn how they work. If any function is used accessing the TCOSS or TC/Archive Server and no TCOSS user ID / password has been configured, you will get a prompt to provide a user ID and password as shown in the screenshot below:




Enter a valid TCOSS user ID and password. The browser will remember it until it is closed.

A very good test to check if the connection the TCOSS server is working is the function as shown in the screen shot below.




```
<?xml version="1.0" encoding="UTF-8" ?>
<c:cl_time xmlns:c="http://www.topcall.com/XMLSchema/2004/tcsi">2009-03-
  17T16:06:23</c:cl_time>
```

Chapter 5

# Application Integration

You can access to TCOSS/TCARCHIVE in your application by using one of the two methods:

- Use simulated web service calls
- Use the web service interface (SOAP)

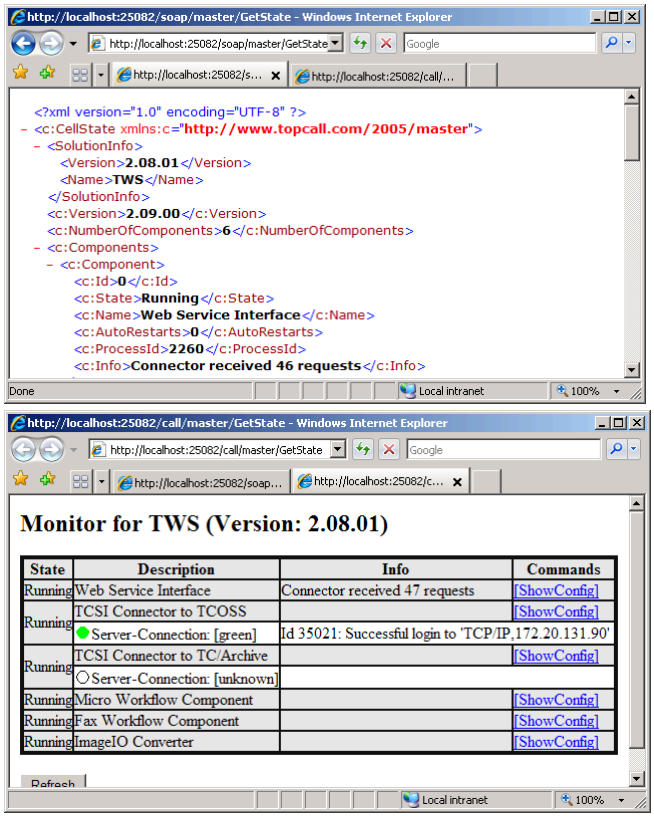
The first method does not require any tools. You just need a web browser. As an introduction, some samples are given on the TWS web portal. The second option can be used to enhance any application (or e.g. workflow system) with access to TCOSS/TCARCHIVE.

## Simulated Web Service Calls

When using this method, the client uses the hyperlink to specify the request. The following formats are supported:

| URL example | Description |
|---|---|
| {host:port}/soap/[target}/{fun} | Call function {fun} in {target} on {host}<br>**Example**: http://localhost:25082/soap/master/GetState |
| {host:port}/soap/[target}/{fun}?{Parameters} | Call function {fun} with parameters {Parameters} in {target} on {host}. Multiple values have to be separated with “&” characters. Complex (nested) parameters are not supported.<br>**Example**: http://localhost:25082/soap/master/Stop?Id=1 |
| {host:port}/soap/[target}?{XML Request} | Call function {XML Request} in {target} on {host}.<br>**Example**: http://localhost:25082/soap/master/<Stop><Id>1</Id></Stop> |
| {host:port}/soap/[target}/{fun}?{XML Parameters} | Call function {fun} with parameters {XML Parameter} in {target} on {host}<br>**Example**: http://localhost:25082/soap/master/Stop?<Id>1</Id> |

All formats described above will return the result of the call as XML data shown as tree structure in the web browser. If you use {host}/call/ instead of {host}/soap, the result will be formatted either via style-sheet or by calling the associated application (e.g. if a binary data block is returned). If formatting information is not available, the data will be shown as XML. You can see the differences in the screen shot below:




## Support of HTML Forms

If you want to enter some variable information into a HTML form, you can use one of following methods:

- Use a java script code that calls an URL according to form fields.
- Send to form data using the form post action.

### Use a Java Script Code

A Java-Script code is used to generate the URL according to the form parameters.

A sample of a simple form to start/stop TCOSS channel is shown below. Note that http://tws-computer in function window.open has to be changed to the actual computer name (and optional port) where TWS is running:

```
<html>
<head>
<meta http-equiv=”Content-Type” content=”text/html; charset=UTF-8”>
<title>Set TCOSS Server Time</title>
<script type=”text/javascript”>
  function SetState(newState) {
    /* open in new window */
 ...  window.open(“http://tws-computer/call/tcoss/set_channel_status?ts_channel=”+
 ...              document.forms[0].Channel.value+”&int_activity=”+newState,
 ...              “_blank”, “width=500,height=200”);
  }
</script>
</head>
```

```
<body>
<form method="GET">
...<p>Enter a valid TCOSS channel number and then press Start/Stop:</p>
  <input type="text" name="Channel" size="2" id="Channel" value="00"/>
  <input type="button" value="Start" onClick="JavaScript:SetState('49')" />
  <input type="button" value="Stop"  onClick="JavaScript:SetState('48')" />
</form>
</body>
</html>
```

This sample can be found in the TWS web portal. Note that Java-Scripts must be enabled in the web browser.

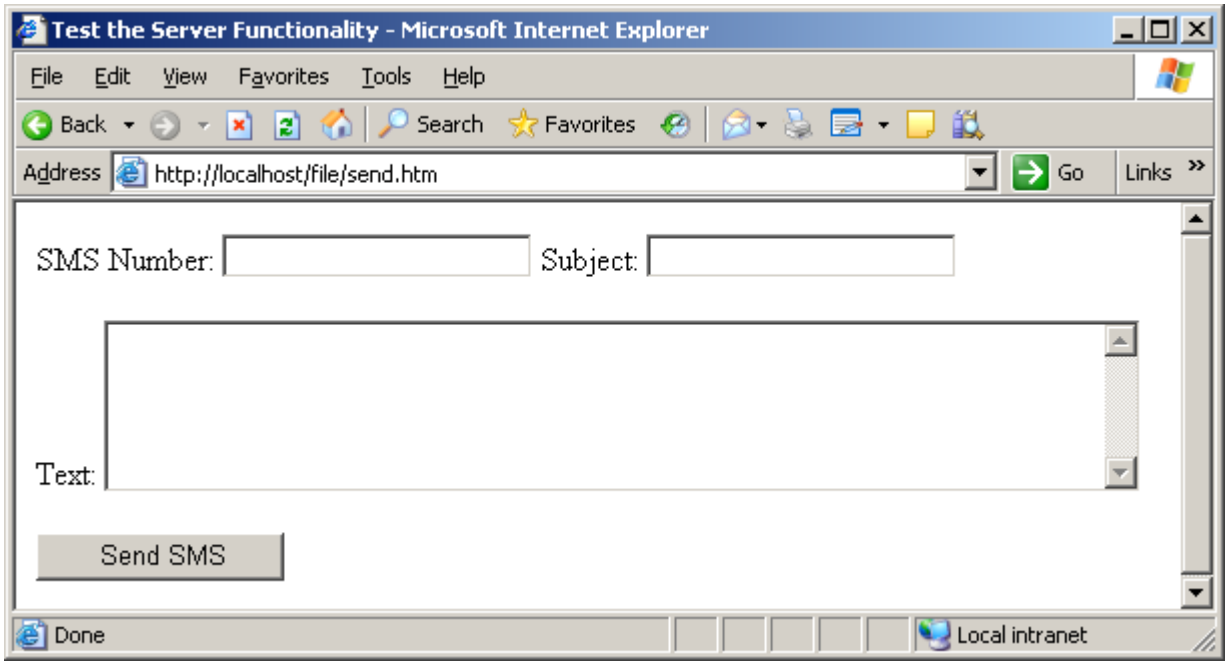
## Use the Form Post Action

In that case, a form must be defined. The name of the input fields must match with the parameter names for each function. If any parameter is to be set to a fixed value, you can use the input element with the attribute type="hidden" (see sample below). The action attribute of the form element must specify the URL to the function that should be called.

The example below can be included anywhere within an HTML page. It shows a simple form that enables the web page to send an SMS. Note that http://tws-computer has to be changed to the actual computer name (and optional port) where TWS is running:

```
<form method="POST" action="http://tws-computer/call/tsl/SendSimpleMessage">
 ...<input name="Service" value="SMS" type="hidden" />
 ...<p>SMS Number: <input name="Number" /> Subject: <input name="Subject" /></p>
 ...<p>Text: <textarea rows="5" cols="62" name="Text"></textarea></p>
 ...<input type="submit" value="   Send SMS  " />
</form>
```

A screenshot of this code (if it is embedded into an empty HTML page) is shown below:



A similar example to send messages with the services FAX, SMS, Voice and Free Number is included in the TWS web portal.

### SMS Sending Limit

Using TWS, you can send maximum up to 5000 SMS with a unique web service call. This performance is measured on a system with the following configuration.

| Test System Information | Description |
|---|---|
| Operating System | Microsoft Windows Server 2012 R2 (64-bit) |
| RAM | 4 GB |
| Processor | Intel® Xenon® 2.50GHZ |

Performance results of a sample application sending messages to ‘n’ recipients and time taken (in seconds). Where, ‘n’ is the number of recipients.

| Number of Recipients (n) | Time taken (in sec) |
|---|---|
| 100 | 3 |
| 200 | 4 |
| 500 | 6 |
| 1000 | 9 |
| 2000 | 14 |
| 5000 | 32 |

**Note** The time for the web service call was measured only and not the actual sending of the messages.

## Web Browser Settings

Some simulated web service calls return a PDF document instead of XML. This may cause problems with the standard settings of web browser.

To start working with Mozilla Firefox start Firefox, select “Tools” – “Options” – “Downloads”, click Plugins… and uncheck (disable) the PDF plugin. When opening PDF the next time, the “opening pfd” window will appear, select “Open with” and look up your PDF viewer. Check the option “Do this automatically for files like this from now on”.

To start working with the Internet Explorer, if you are using Adobe Reader, start the Reader, select “Edit” – “Preferences” and uncheck all options in the “Internet” tab, e.g. uncheck “Display PDF in browser”.

**Note** The Kofax Communication Server offers an open interface that enables the integration of third party application for format conversion (e.g. PDF) but Kofax is not responsible for support and licensing of any product that has not been purchased from Kofax.

# Using the Web Service Interface

In that case, you will find any tool that supports using web services. The concrete procedure depends on the used tool but in general, the following procedure will be common to all tools.

- Specify the location of the WSDL file to define the interface. The following WSDL files are provided by TWS. More details about all available functions can be found in chapter Example with PocketSOAP:

| | |
|---|---|
| {host} /file/tsl.wsdl | => All micro workflow functions |
| {host} /file/tsl1.wsdl | => Micro workflows that do not use the TCSI schema. |
| {host} /file/tcoss.wsdl | => TCOSS Server functions |
| {host} /file/archive.wsdl | => Archive Server functions |
| {host} /file/master.wsdl | => TWS maintenance functions |

The tool will then generate source code to use the web services.

**Note** TWS supports only the document/literal wrapped binding style. A good overview about binding all binding styles can be found in: http://www-128.ibm.com/developerworks/webservices/library/ws-whichwsdl/. Applications using TWS must comply with SOAP standard 1.1.

**Note on timeout settings**: TWS uses internal timeout values in a range of 1 – 2 minutes. If your SOAP client allows to set the timeout it employs in waiting for the web service response, it is recommended to use a timeout value of 2 minutes to make sure that the response is received.

The following WEB-Service clients have been tested with TWS:

- Visual Studio .net 2003 (VB and C#):
  Tests have been done with http and https transport
- Java (Development Kit 1.5 and Netbeans 4.1)
  Tests have been done only with http transport.
- Perl (tested with ActivePerl 5.8.4.810 – see http://www.ActiveState.com/ActivePerl/)
  Tests have been done only with http transport.
- PocketSOAP (tested with version 1.5.3 and VB-Script – see http://www.pocketsoap.com)
  Test have been done with http and https transport.

If you want to write your own application using the TWS Web Services it is recommended to take a look into the example programs that are available. These examples are described in the next chapter.

Chapter 6

# Code Example Using TWS

All examples described in this chapter are installed with TWS as zip-archive in this folder: C:\TOPCALL \TWS\00\Samples.

The table below gives you a short overview about the examples.

| Sub-Directory | Short description | See |
|---|---|---|
| DotNet\CSharp-Simple \SendSimpleMessage | Simple C# console application that sends a text message | Send a Simple Text Message |
| DotNet\CSharp-Simple\GetState | Simple C# console application that returns the current state of TWS | Get State of TWS |
| DotNet\CSharp | Tws-Examples as console applications for SendMessage, ReceiveMessage, ViewMessage, TrackMessage, CancelMessage, GetMessageEntry and ReactiveateMessage | TWS Examples |
| DotNet\VB\SendSimpleMsg | VB form that supports sending of a simple text message. It supports HTTPS, user authentication. | Send Simple Message |
| DotNet\VB\tcChannelStatus | VB form that shows the current state of TCOSS channels. Channels can be started or stopped. It supports https, user authentication. | Show and Change TCOSS Channel State |
| DotNet\VB\ReceivePDF | VB form that supports reception of messages from any queue as PDF file. It supports https, user authentication and a connection via proxy server. | ReceivePDF |
| Java\GetTime | Simple Java code to read the current TCOSS time. | Example with Java |
| Perl-soaplite | Some simple Perl Script examples. (e.g. read the current TCOSS time) | Examples with Perl |
| VBScript-PocketSoap | Several samples with VB-Script that uses the COM object PocketSOAP | Example with PocketSOAP |

## Example with Visual Studio .net (C#)

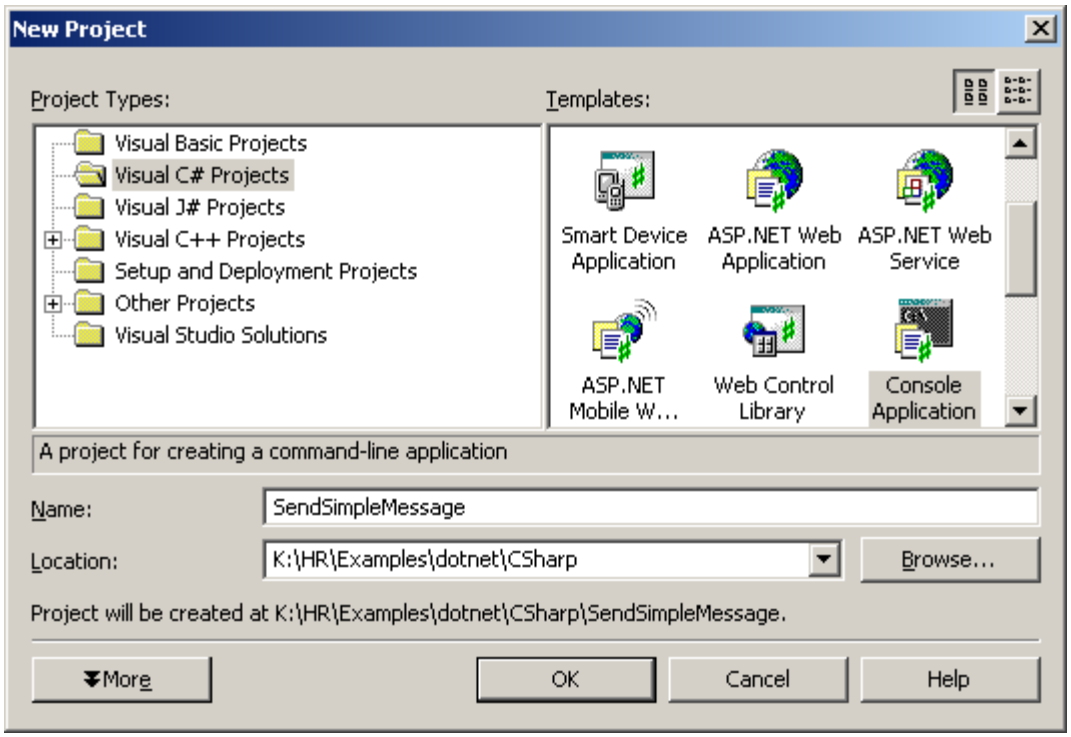
This chapter describes some examples that were made with Visual Studio .net

### Send a Simple Text Message

The following example explains step by step how to write a simple C# program that uses a TWS function to send a simple message. It assumes that the TCOSS User Id and Password is configured in the TWS Server.

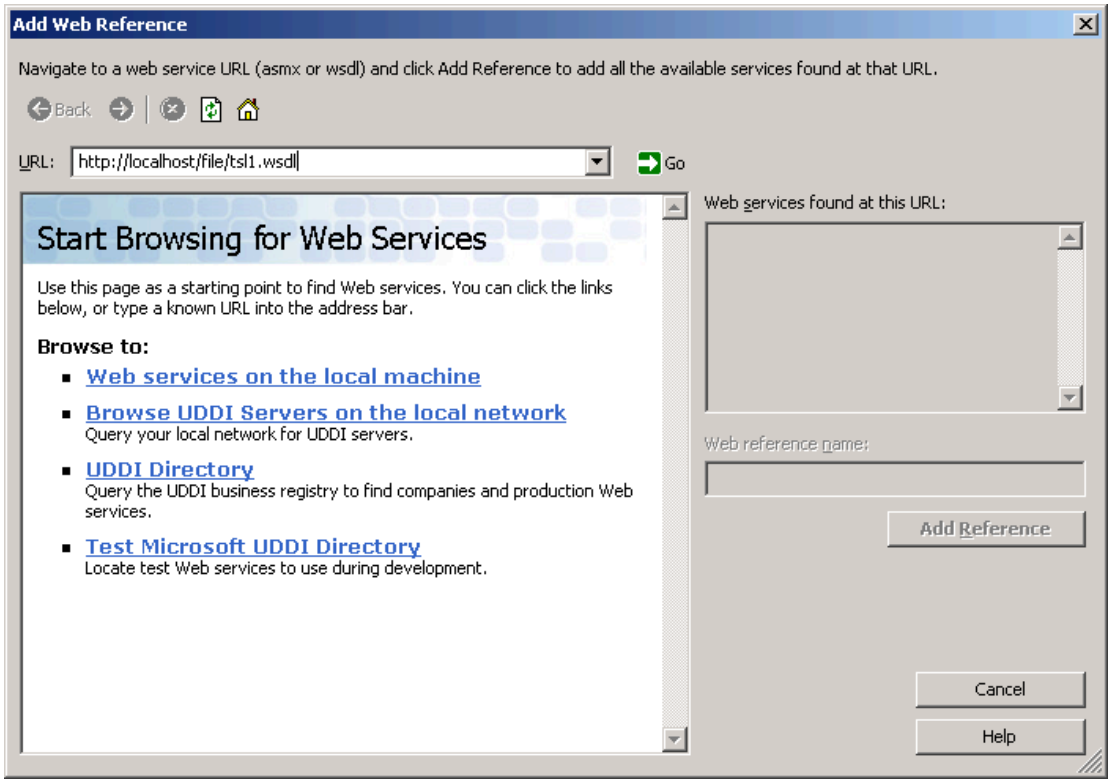
If you want to use basic authentication or https, please refer to the Visual Studio .net samples for Visual Basic. Since both languages are based in the .net common language runtime, the VB implementation can be easy ported to C#.

## Step 1: Create a New C# Console Application

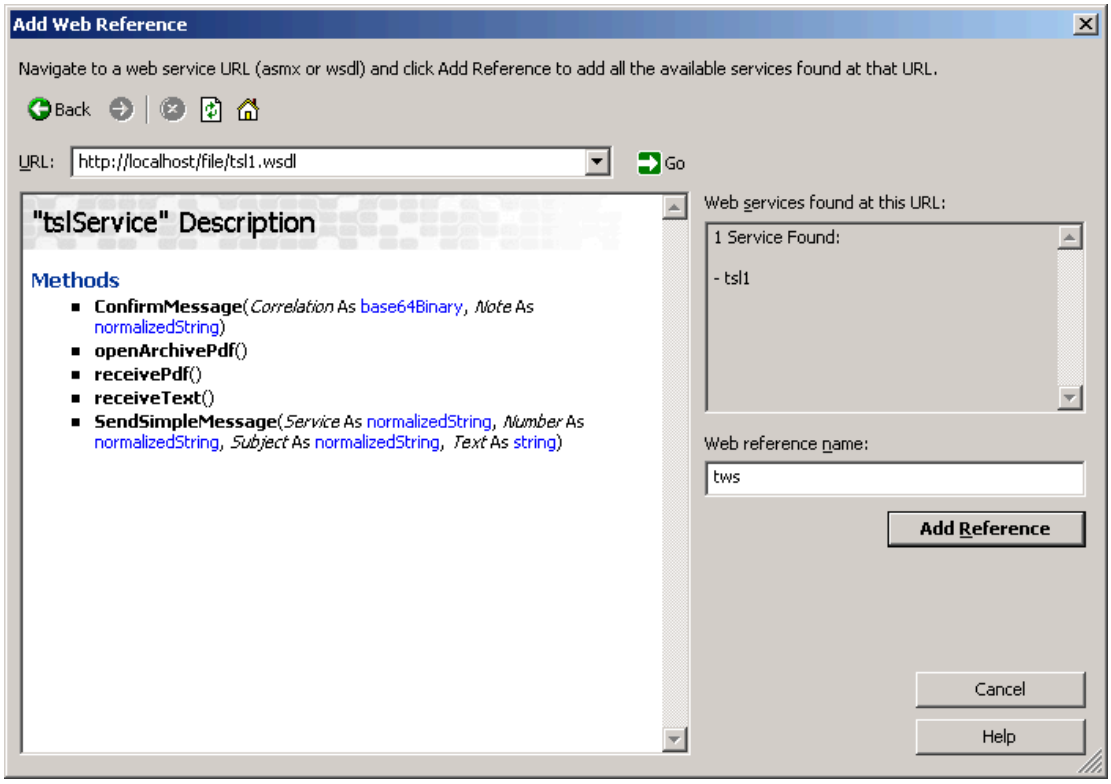



## Step 2: Add the Web Reference

Select „Add Web reference“ (e.g. via menu “Project -> Add Web Reference...”) and enter the URL of the WSDL file (e.g. http://localhost/file/tsl1.wsdl if TWS is running on your local machine).




Click “Go”. You should get an overview about all web service calls defined in tsl1.wsdl. It is recommended to change the web reference name from the host name (which is default) to any meaningful name that is unique within your project. In this example, the name was changed to “tws”. The result is shown in the screen shot below.




## Step 3: Create Code to Call the Web Service

The following code sample sends message “This is a test” with subject “test” to KCS user „TCTECH“.

```
using System;
namespace SendSimpleMessage
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
            try
            {
                tws.tslService tsl = new tws.tslService();
                tws.SendSimpleMessage par = new tws.SendSimpleMessage();
                par.Service = “FREE”;
                par.Number = “TCTECH:”;
                par.Subject = “test”;
                par.Text = “This is a test”;
                tsl.SendSimpleMessage(par);
                Console.WriteLine(“Message has been sent\n”);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.ToString());
            }
        }
    }
}
```

## Pitfalls

This chapter gives you some hints if the program above does not work.

## The Host Name Is Correctly Configured

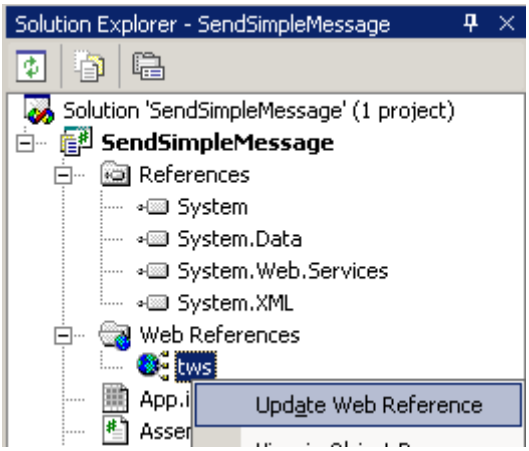
If there has not been configured any correct host name (or none at all) the program above will fail with an error as shown below:

```
System.UriFormatException: Invalid URI: The hostname could not be parsed.
   at System.Uri.Parse()
   at System.Uri..ctor(String uriString, Boolean dontEscape)
   at System.Uri..ctor(String uriString)
   at System.Web.Services.Protocols.WebClientProtocol.set_Url(String value)
   at SendSimpleMessage.tws.tslService..ctor() in ...\tws\reference.cs:line 31
   at SendSimpleMessage.Class1.Main(String[] args) in ...\class1.cs:line 11
```

**How to solve the problem**:

Run the TWS configuration and check the correct configuration of the host name.


After the configuration has been corrected the web service reference must be refreshed. In that case the Web Reference must be updated as shown in the screen shot below.




## No KCS User ID / Password Was Configured

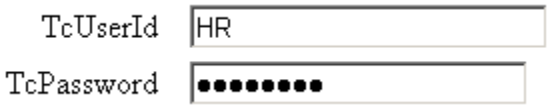
If using the example described above, the TCOSS user ID and password must be configured in the TWS. If you do not use a predefined user ID / password, the program will fail with an error as shown below:

```
 System.Net.WebException: The request failed with HTTP status 401: Unauthorized.
   at
System.Web.Services.Protocols.SoapHttpClientProtocol.ReadResponse(SoapClientMessage
 message,
WebResponse response, Stream responseStream, Boolean asyncCall)
```

```
   at System.Web.Services.Protocols.SoapHttpClientProtocol.Invoke(String methodName,
Object[] parameters)
   at SendSimpleMessage.tws.tslService.SendSimpleMessage(SendSimpleMessage
SendSimpleMessage1) in D:\src\server\TCOSS8\samples\SendSimpleMessage\Web References
\tws\
Reference.cs:line 59
   at SendSimpleMessage.Class1.Main(String[] args) in
D:\src\server\TCOSS8\samples\SendSimpleMessage\Class1.cs:line 17
```

### How to solve the problem

1. Run the TWS configuration and configure a correct TCOSS user Id or password. You must restart TWS in order to use the new configuration.




2. Use a web browser and check if you can send a message using the TWS web portal.

## Get State of TWS

This example uses the web service call to get the current state information of TWS. Most things are the same as described above in the example Send a Simple Text Message. The main difference is that this project requires the file master.wsdl (instead of tsl1.wsdl). It demonstrates how to process a response of web service function. The required source code is shown below:

```
using System;

namespace GetState
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {

      tws.masterService tsl = new tws.masterService();
      Console.WriteLine("Get TWS state from url={0}", tsl.Url);
      tws.GetState par = new tws.GetState();
      tws.CellState Resp = tsl.GetState(par);
      Console.WriteLine("Response Number Of Components {0}",
Resp.NumberOfComponents);
      for (int i=0; i < Resp.Components.Length; i++)
      {
        Console.WriteLine(" {0} - {1} - {2}",
               Resp.Components[i].State,
               Resp.Components[i].Name,
               Resp.Components[i].Info);
      }
      Console.WriteLine("");
    }
  }
}
```

## TWS Examples

The solution file “Tws-Examples.sln” consists of a collection of .Net C# console application that use the TWS functions SendMessage, ReceiveMessage, ViewMessage, TrackMessage, CancelMessage, GetMessageEntry and ReactiveateMessage.

In the folder “DotNet\CSharp\Export” you find the executables plus the configuration files. The configuration files show what has to be configured to call the programs. They have to be located in the same folder as the executables.

Please note the following:

- All programs depend on “ViewMessage.exe”; so to use them, “ViewMessage.exe” has to be in the same folder.
- ReceiveMessage and ViewMessage implement only one set of parameters for ImageView, SelectType and Representation.

# Example with Visual Studio .net (VB)

This section contains example with Visual Studio .Net (VB).

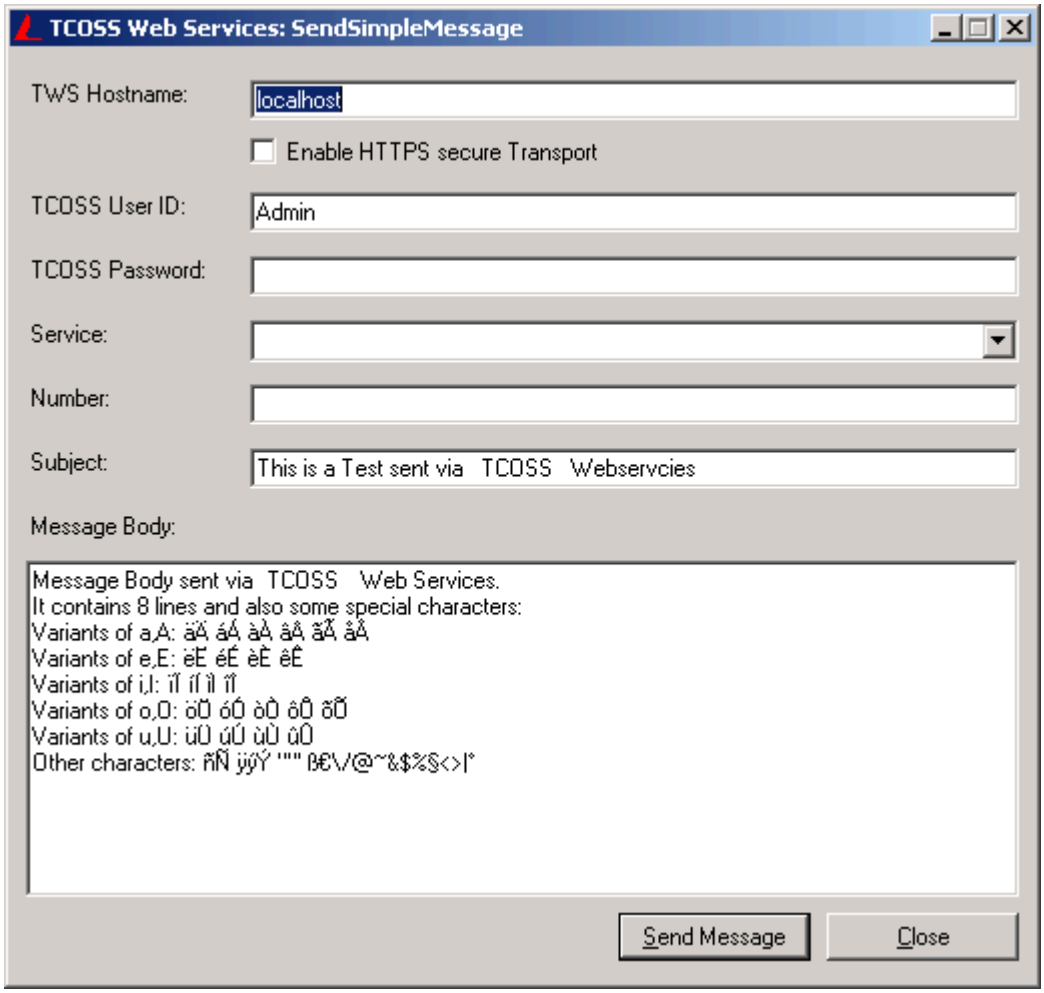
## Send Simple Message

This is a VB form that supports sending of a simple text message. It supports http or https and user authentication. A screen shot is shown below:

Please note, that the Microworkflow for sending a simple message only resolves the service prefix but not the service type. That means, when sending e.g. to a KCS user you have to append the “:” after the user ID:

| | |
|---|---|
| Service: | TOPCALL |
| Number: | Admin: |

Some characters of the Message body are not available in the TCOSS Codepage 0. Therefore – when checking the message using TCfW you will not find back all characters entered within the message body.

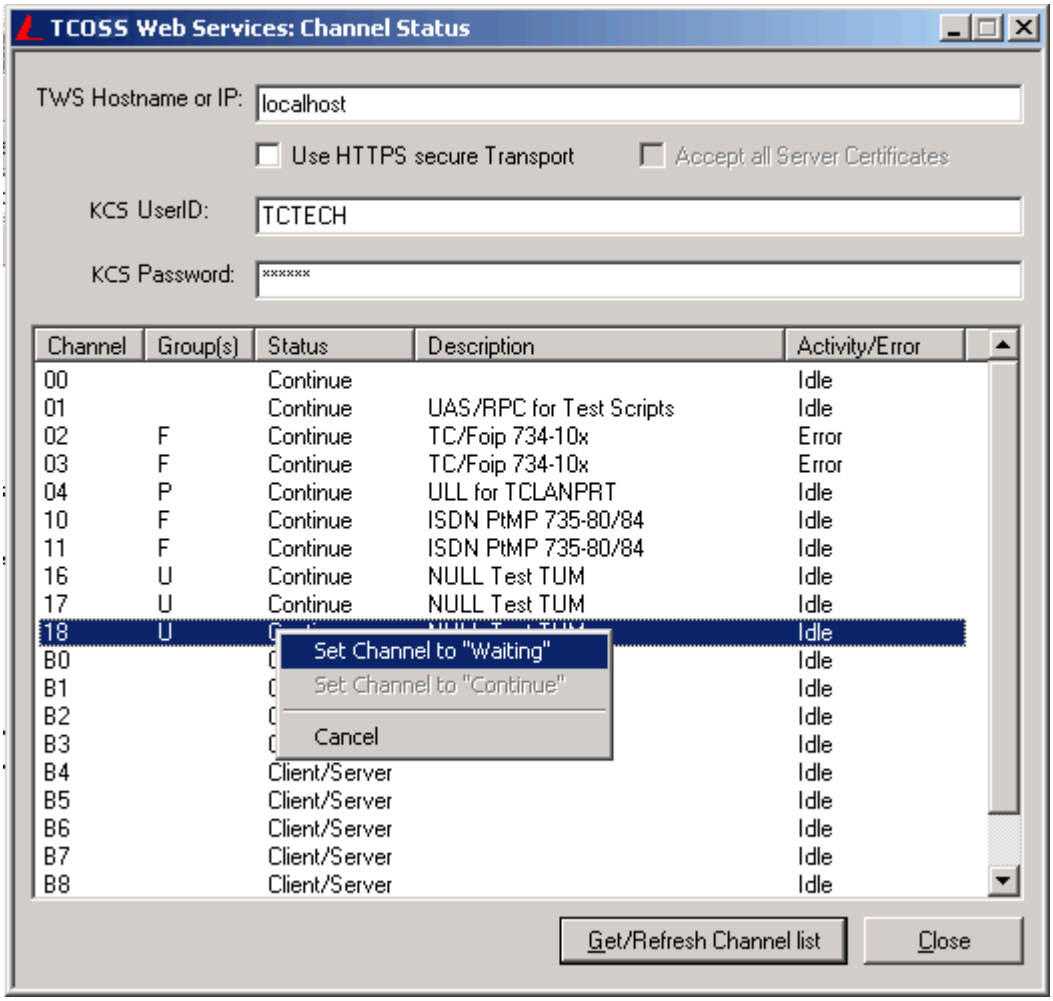



Implementation details can be found in the source code comments.

## Show and Change TCOSS Channel State

This is a VB form that shows the current state of TCOSS channels. Channels can be started or stopped. It supports HTTP or HTTPS and user authentication.

A screen shot is shown below:




Implementation details can be found in the source code comments.

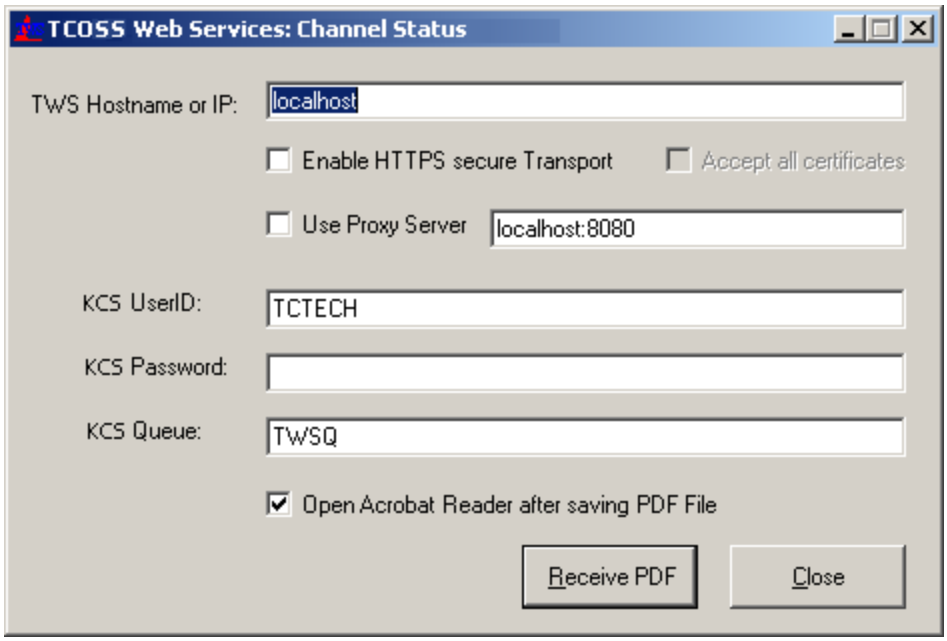
Remarks:

- If the check-box “Accept All Server Certificates” is checked, the application accepts all server certificates even if some certificate requirements are not fulfilled.

  For example: The certificate was requested with the hostname of the TWS machine. Now you connect to the TWS machine using https transport but using the IP address instead of the hostname. With a web browser, you would get an alert “The name of the security certificate is invalid or does not match the name of the site” but a web service client would simply fail. If you enable that checkbox, the communication should work anyway.
- The state of a channel can be changed in the context menu. You can only change the status from “continue” to “waiting” and vice versa. The status of “Query” or “Client/Server” channels cannot be changed.

## ReceivePDF

This is a VB form that supports reception of messages from any KCS queue as PDF file. It supports http or https transport, user authentication and configuration of a proxy server (see chapter “HTTP Proxy Tracer” within the “Troubleshooting” of this manual). A screen shot is shown below:




When you press the button “Receive PDF” the program gets the next messages from the KCS Queue and saves it as PDF file into the current directory. The filenames for the PDF Files are chosen automatically. After retrieving the File from the TCOSS Queue, the send order is confirmed to TCOSS. You will see the timestamp, when “Receive PDF” has confirmed the message within the “Response” column of the TCfW outbox.

The KCS Queue (“TWSQ” in the example above) should be defined with option “Visible in Outbox” checked (defined in the general section of the KCS user profile), otherwise the response “Confirmed by TWS: + timestamp” is not stored by TCOSS and not visible in the TCfW outbox.

# Example with Java

This section describes example with Java.

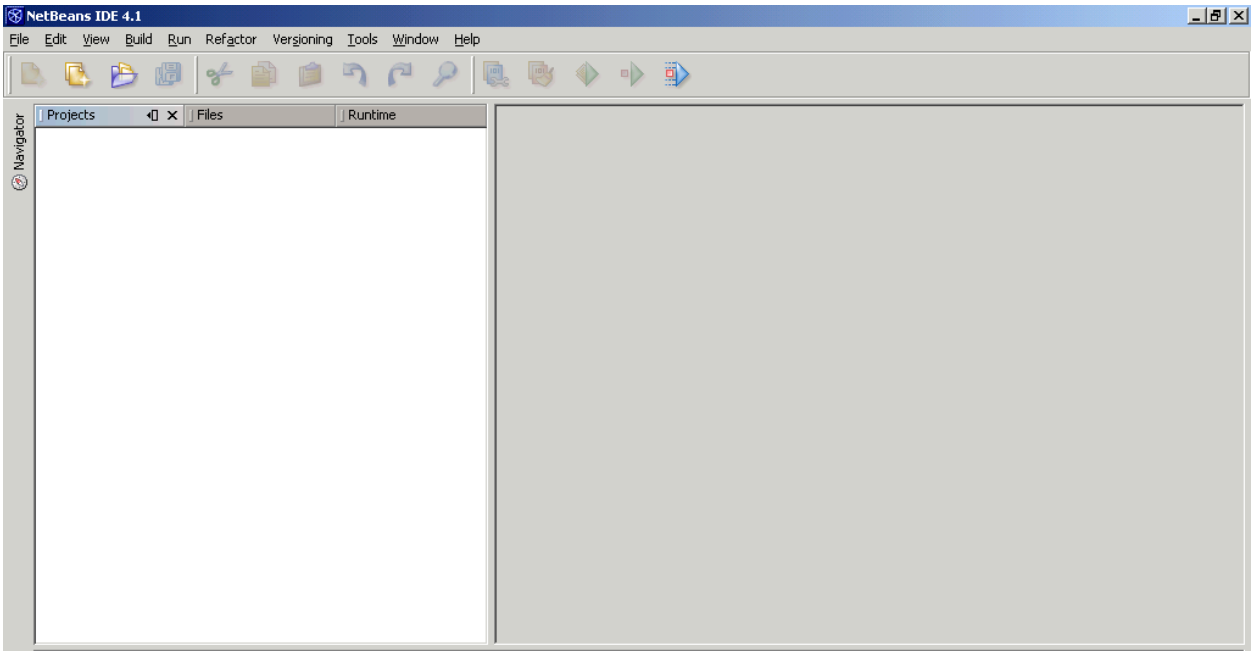
## Prerequisites

- Java Runtime Environment 1.5 (JRE 1.5.0_05)
- Java Development Kit 1.5 (JDK 1.5.0_05)
- Netbeans 4.1 including the SUN – Application Server 8 or
- Netbeans 4.1 and the Java Web Service Development Kit 1.6

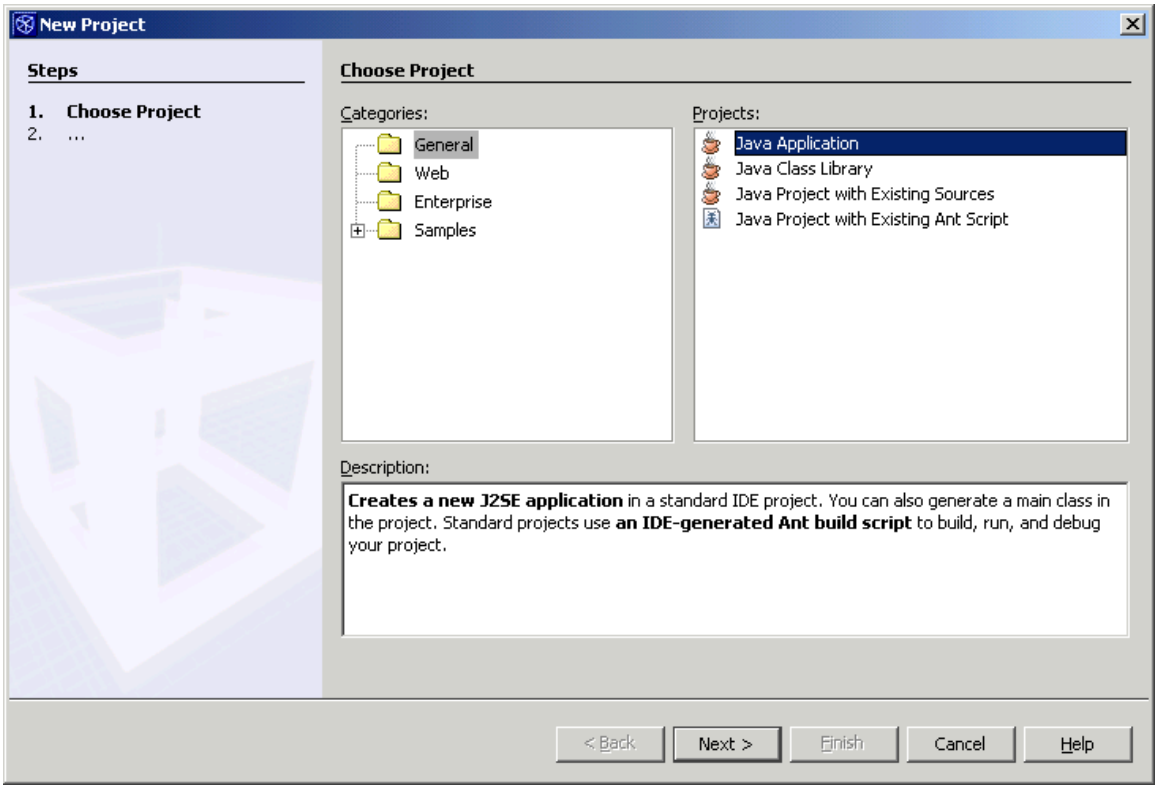
The directory “C:\Program Files\Java\jdk1.5.0_05\bin” must be part of your path environment variable. Depending on what you installed I would suggest also including the directory where the file “wscompile.bat” is located.

## Step 1: Create a New Project

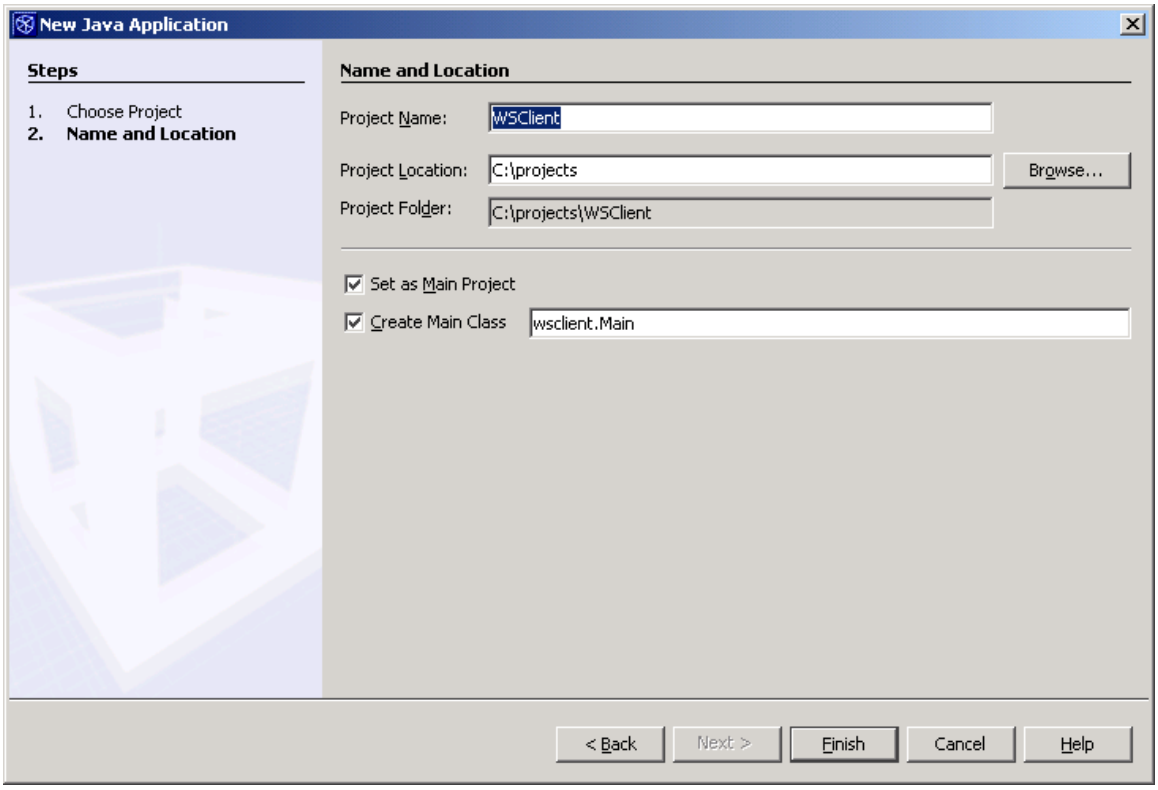
1. Create a new project **File** > **New Project...**



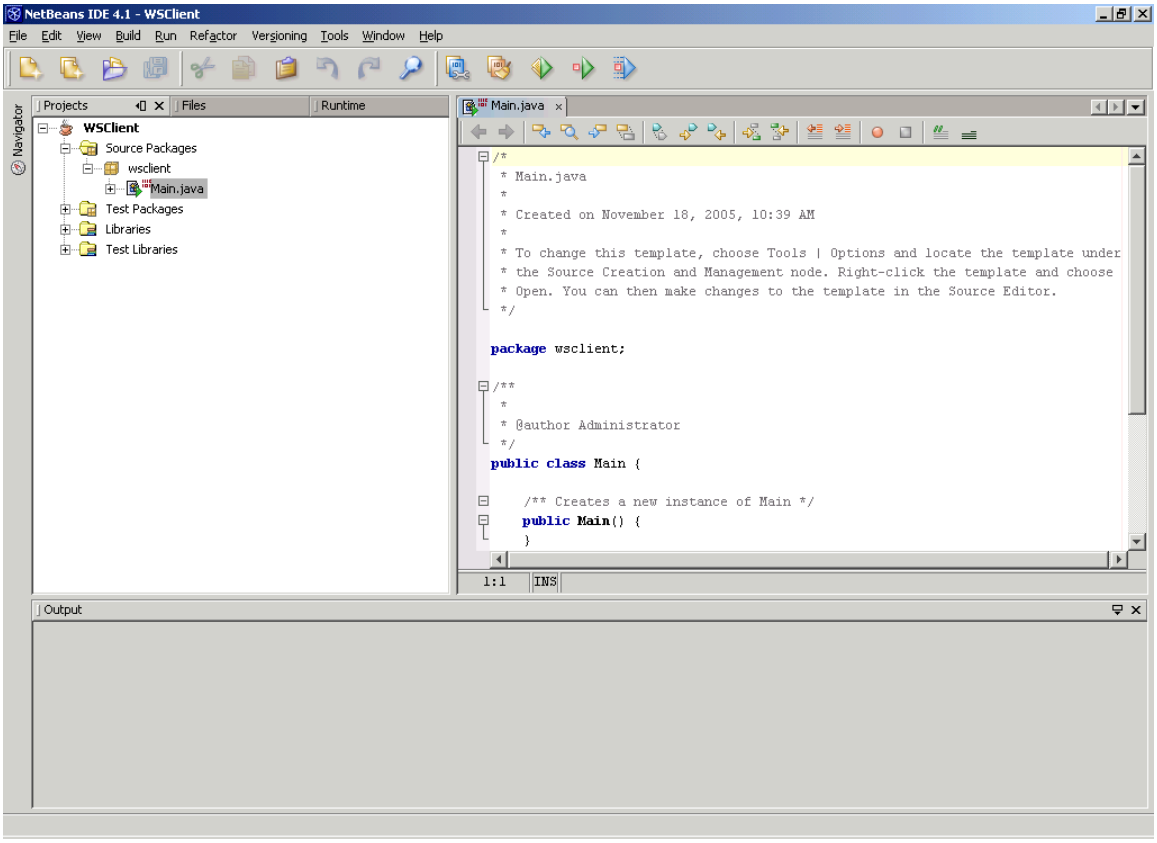
2. Select **Java Application** and click **Next**.



3. Enter a name (“wsclient”) and a location for the new project. Then click **Finish**

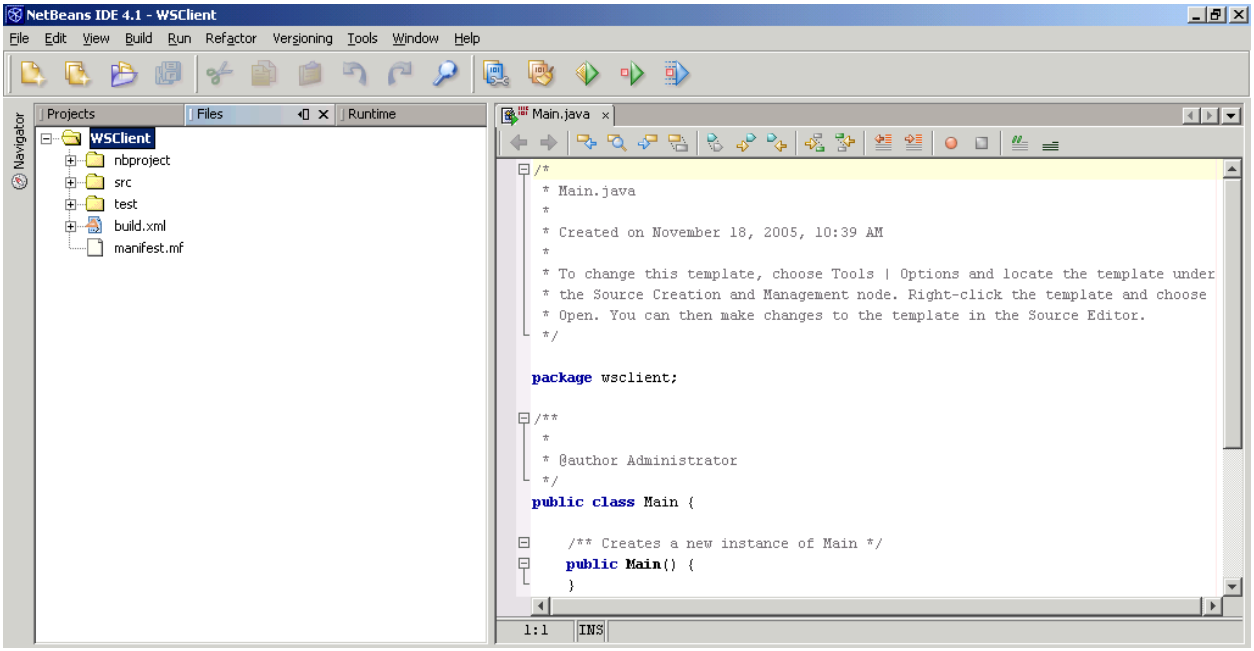


4. You should see something like the following picture

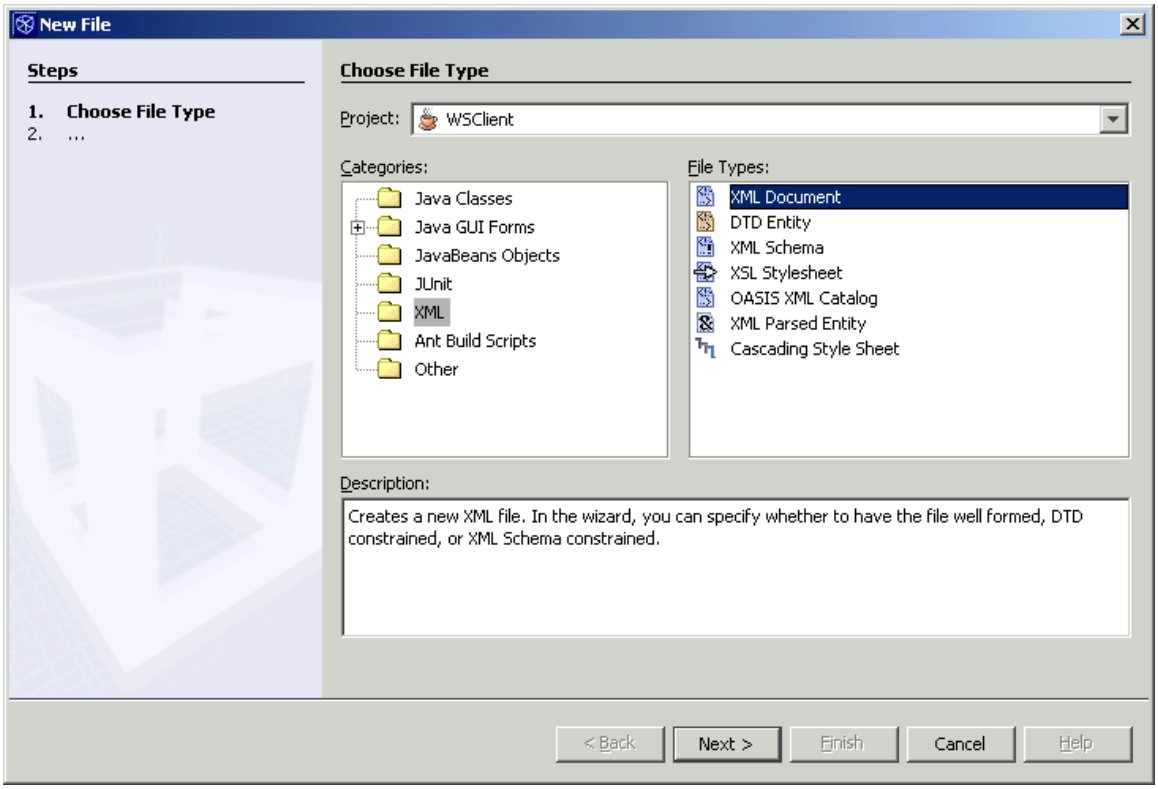


## Step 2: Import the WSDL File

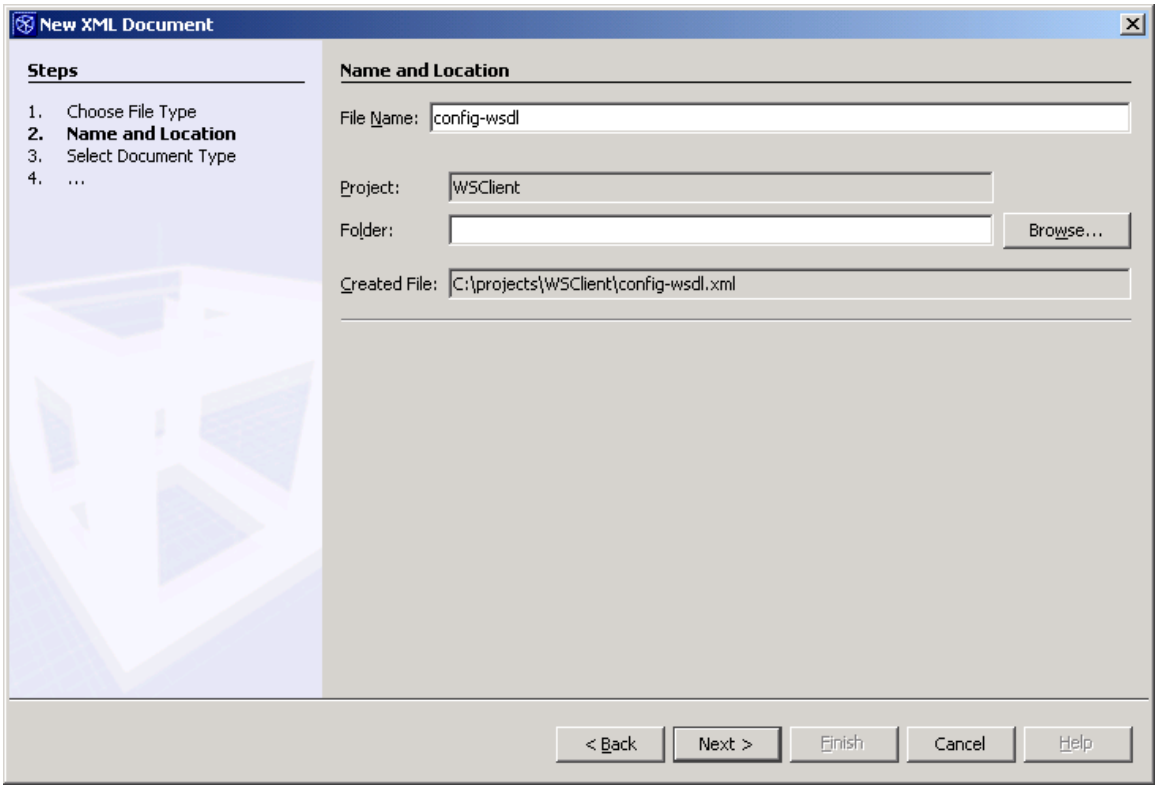
1. Change to the file tab



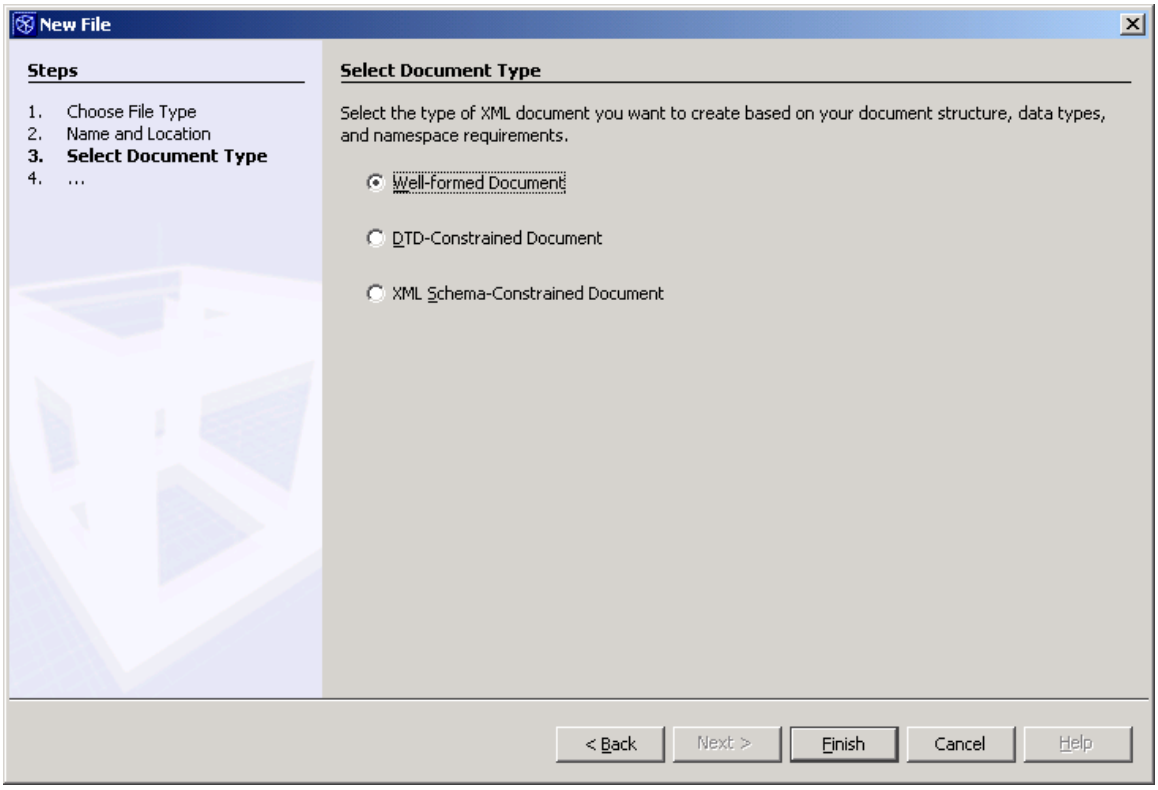
2. Right click on the root directory and choose “New -> File/Folder…” In the new file dialog select “XML – XML Document” and click “Next”.



3. In the following dialog change the name to “config-wsdl.xml” and click “Next”.



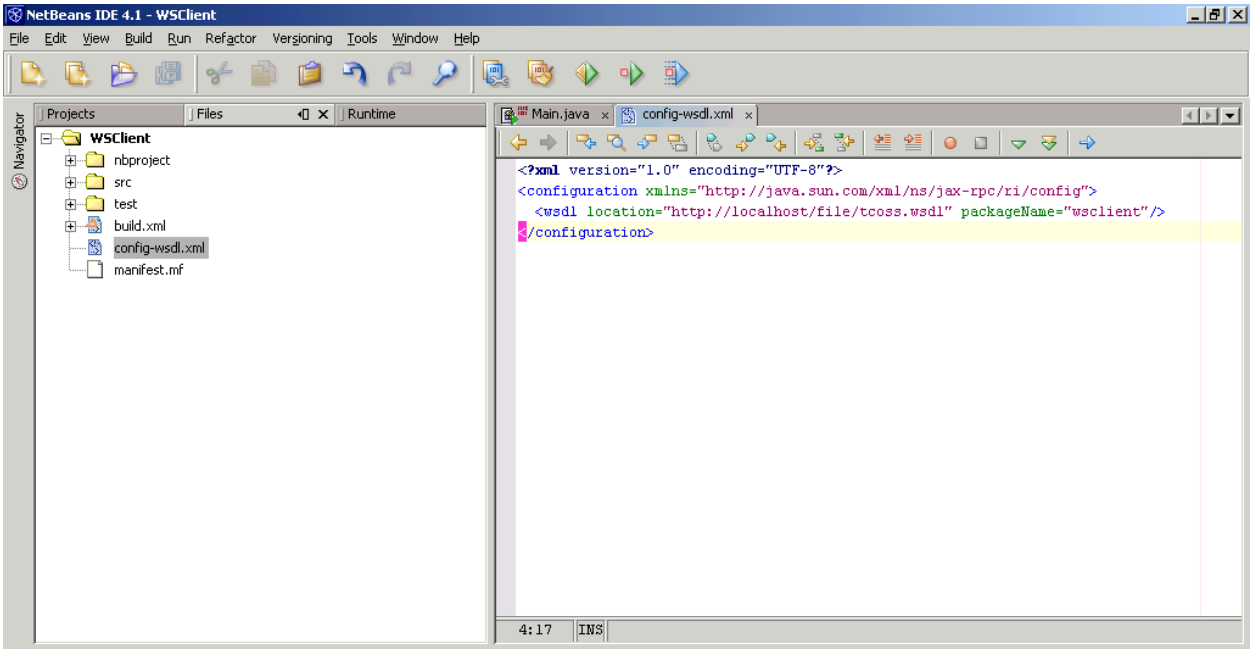
4. At the end select “Well-formed Document” and click “Finish”.



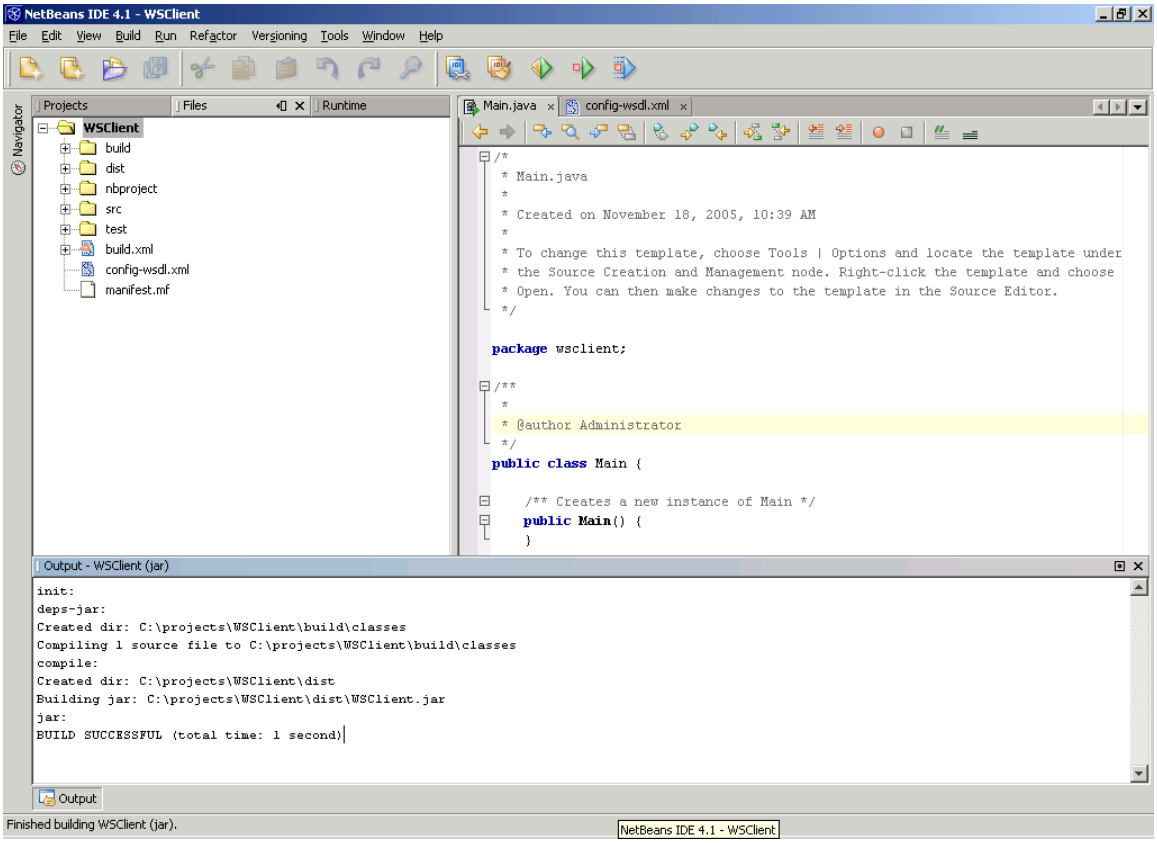

5. Change the text in the “config-wsdl.xml” file to:

```
<?xml version=”1.0” encoding=”UTF-8”?>
<configuration xmlns=”http://java.sun.com/xml/ns/jax-rpc/ri/config”>
<wsdl location=”http://localhost/file/tcoss.wsdl” packageName=”wsclient”/>
</configuration>
```

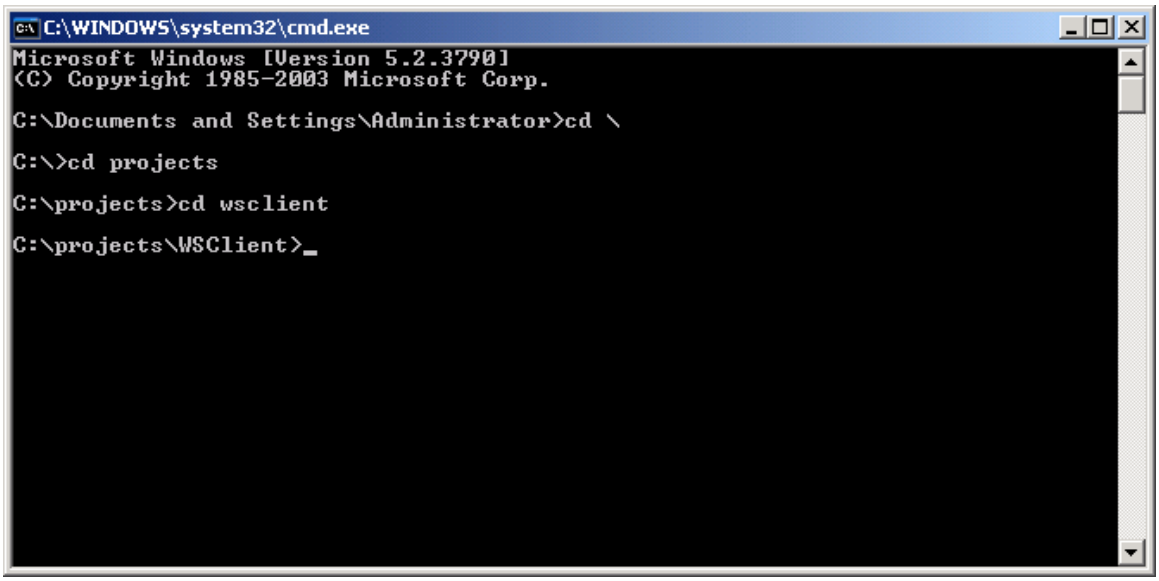
6. Replace “localhost” with the name of the server where TCOSS8 is running. Now it should like this:



7. Now build it so that Netbeans creates the necessary directory structures.



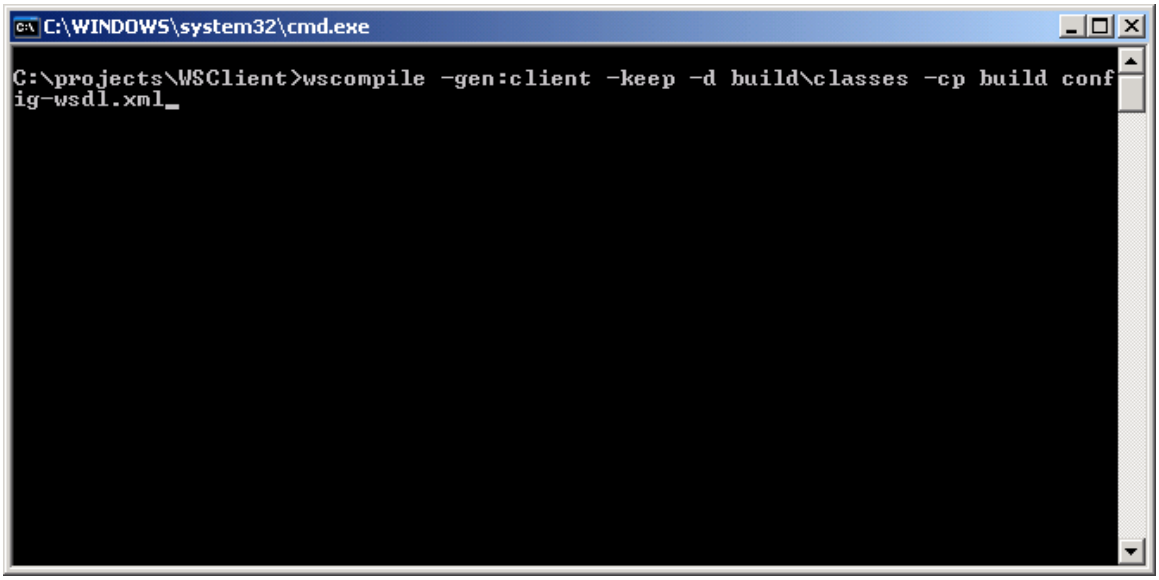

8. Open a command line and change into the projects root directory. (Be sure to set the path environment correct)



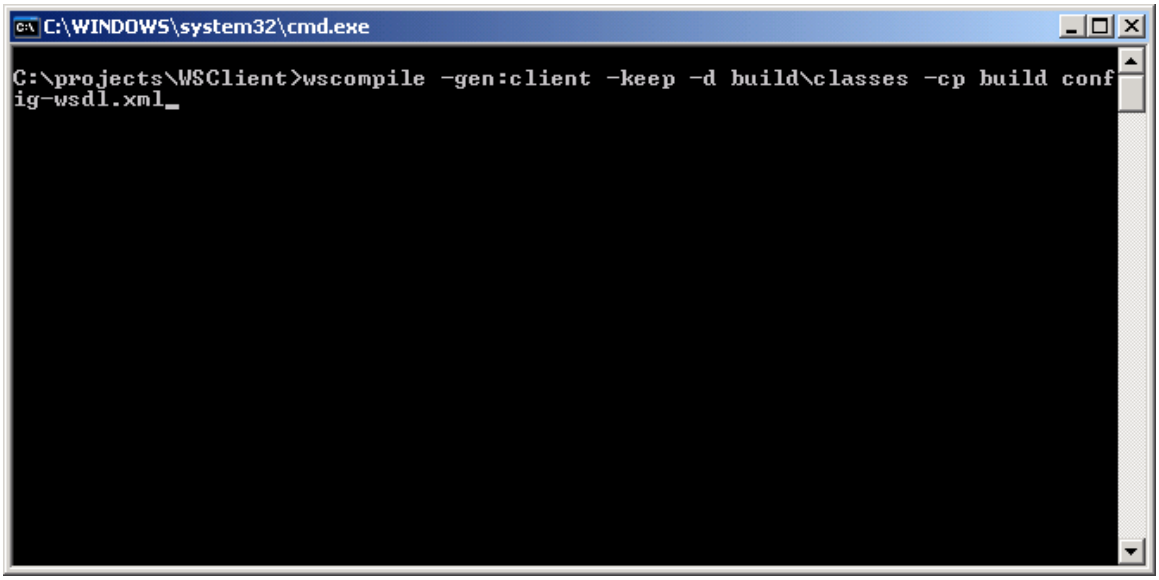
9. Run the following command:

```
wscompile -gen:client -keep -d build\classes -cp build config-wsdl.xml
```

This command will download and parse the WSDL file specified in “config-wsdl.xml”. Then it will create all necessary java classes and files.

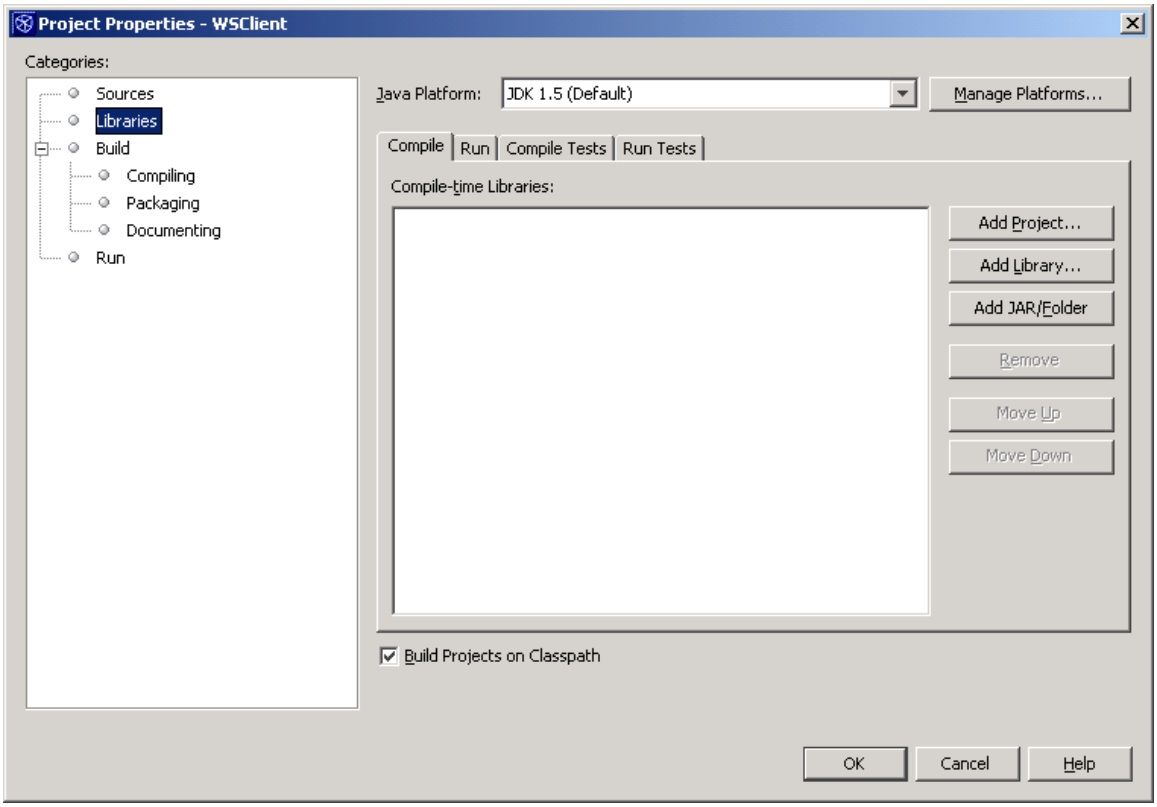



10. Move the generated *.java file from “.\build\classes\wsclient\” to “.\src\wsclient” (This is done to allow a clean – rebuild)



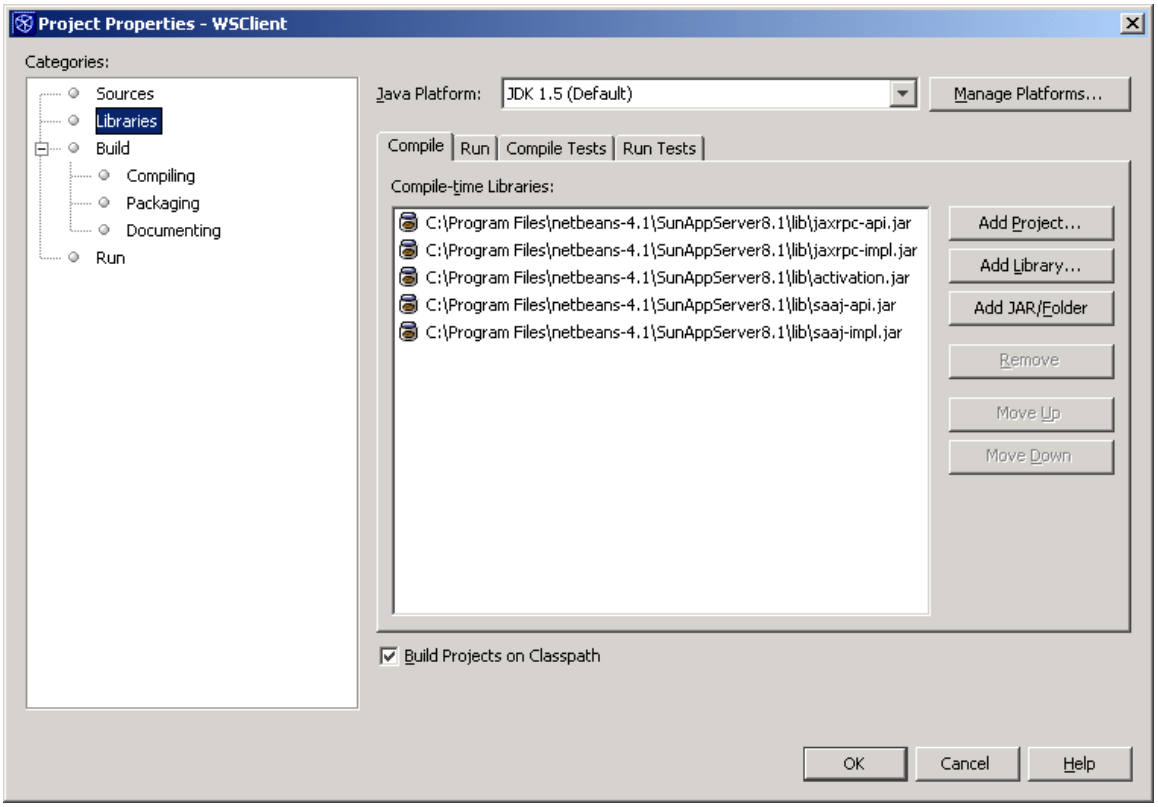

Before proceeding, it is necessary to add additional jar’s to the projects dependency. Select “File -> WSClient properties”.

**11.** Change into the category “Libraries” and open the tab “Compile”:



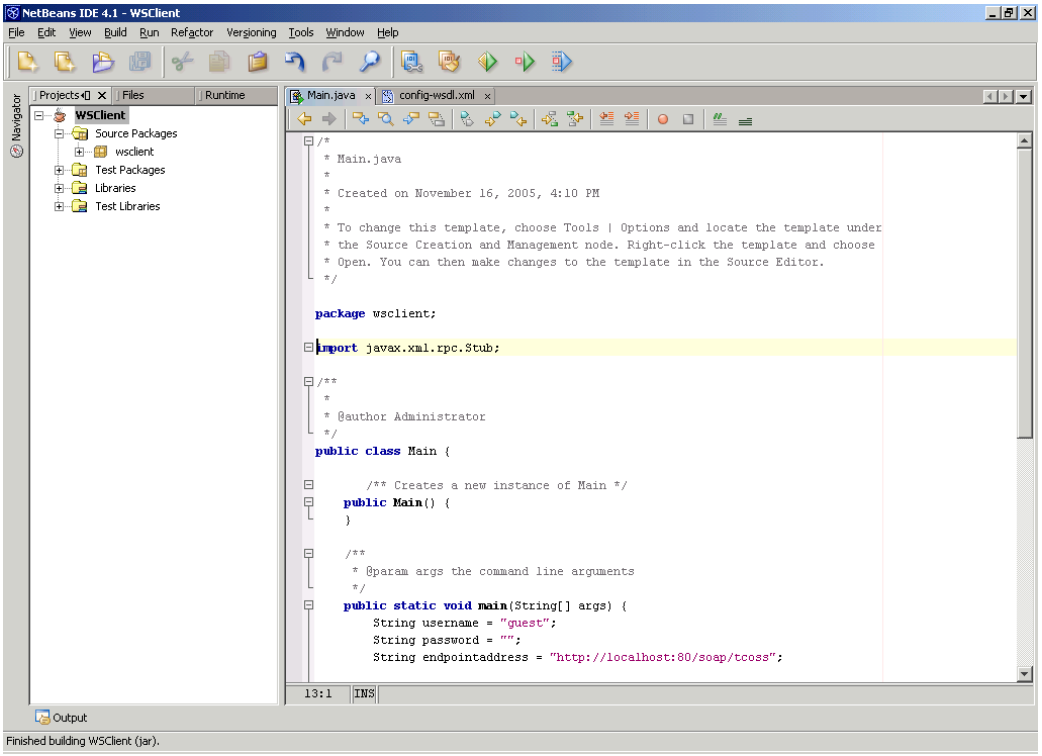
**12.** Add the following jars. The location of the files may vary depending on the chosen installation.

- jaxrpc-api.jar
- jaxrpc-impl.jar
- activation.jar
- saaj-api.jar
- saaj-impl.jar



## Step 3: Create Code to Call the Web Service

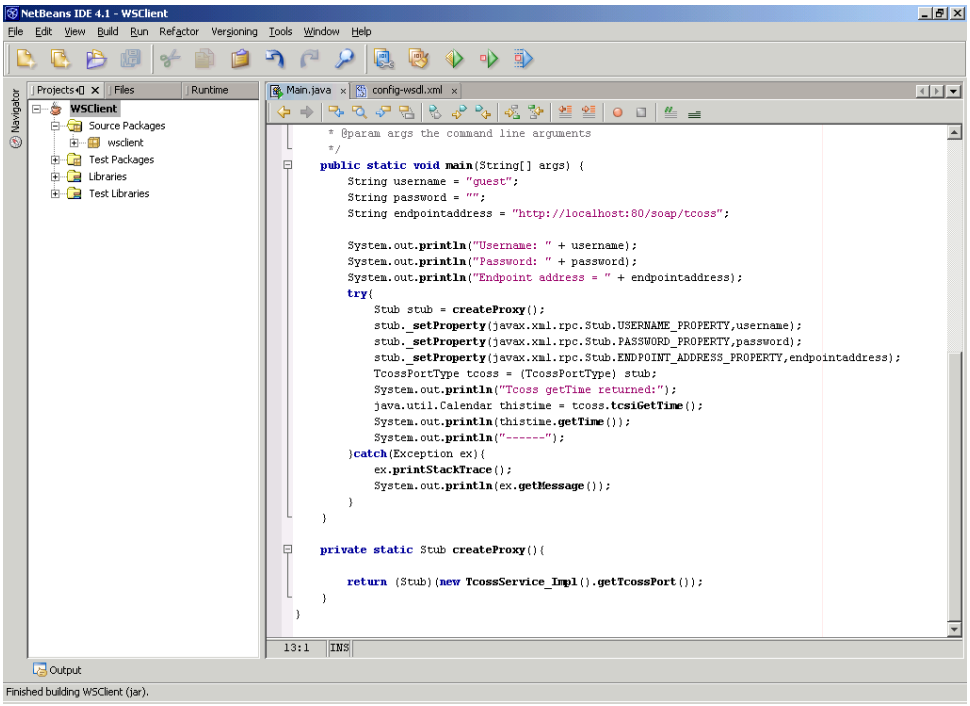
1. Now add the following statement after the package declaration import javax.xml.rpc.Stub;




2. Now fill the main function with the following text:

```
 String username = "guest";
        String password = "";
        String endpointaddress = "http://localhost:80/soap/tcoss";

        System.out.println("Username: " + username);
        System.out.println("Password: " + password);
        System.out.println("Endpoint address = " + endpointaddress);
        try{
            Stub stub = createProxy();
stub._setProperty(javax.xml.rpc.Stub.USERNAME_PROPERTY,username);
 stub._setProperty(javax.xml.rpc.Stub.PASSWORD_PROPERTY,password);
 stub._setProperty(javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY,endpointaddress);
            TcossPortType tcoss = (TcossPortType) stub;
            System.out.println("Tcoss getTime returned:");
            java.util.Calendar thistime = tcoss.tcsiGetTime();
            System.out.println(thistime.getTime());
            System.out.println("------");
        }catch(Exception ex){
            ex.printStackTrace();
            System.out.println(ex.getMessage());
```

```
        }
```

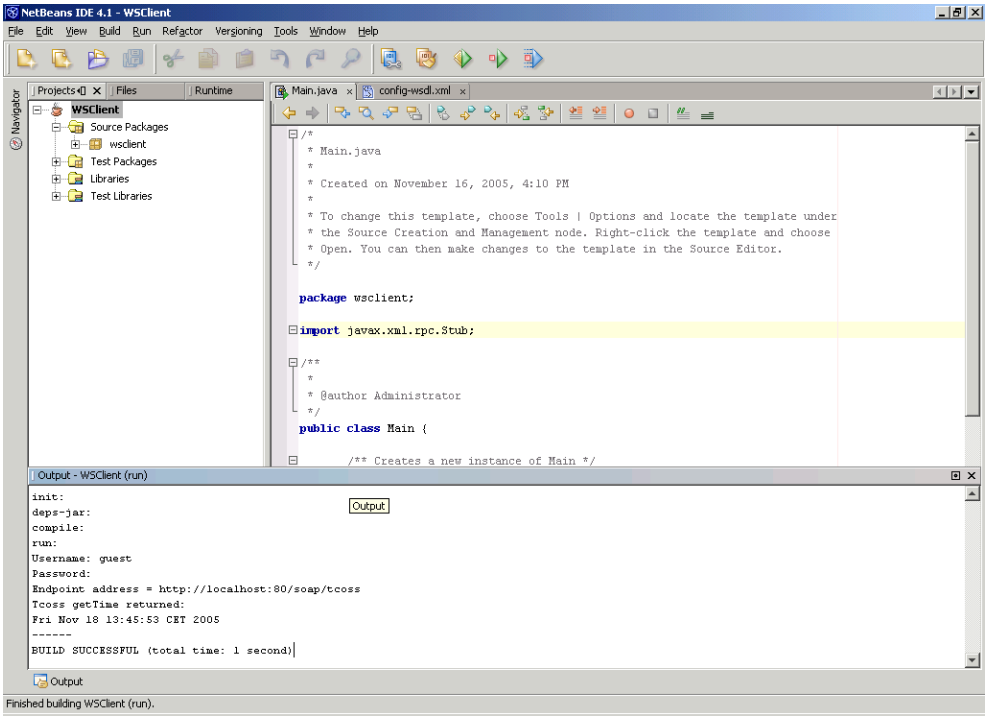



3. And add the following function:

```
private static Stub createProxy(){
        return (Stub)(new TcossService_Impl().getTcossPort());
    }
```

4. Finally replace “localhost” with the name of the server where TCOSS 8 is running and check if the username and password apply.

5. Now run build and run the project.




# Second Example with Java

This example shows how to call the sendFax Web Service with Java. It takes two arguments:

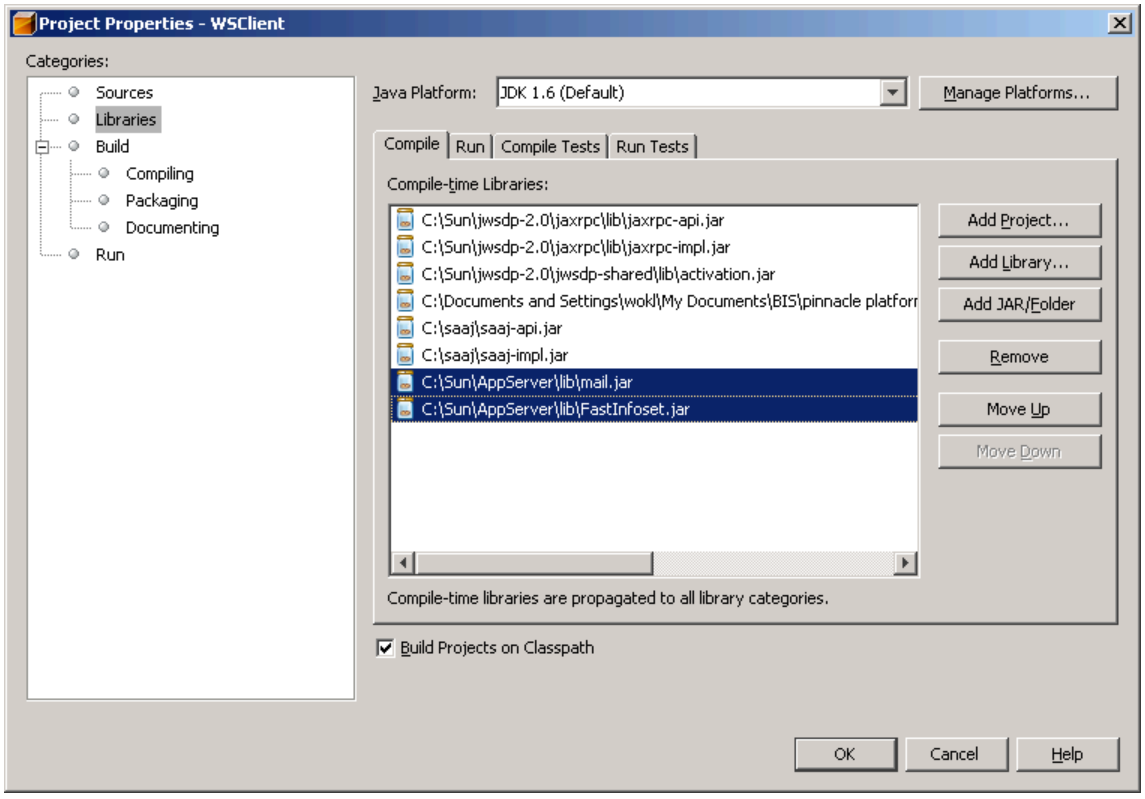
- Filename of the image to be sent
- Number to which the fax is sent

## Prerequisites

In addition to the libraries mentioned in chapter 6.3 the files mail.jar and FastInfoset.jar are needed. This example was created with JDK 1.6 and jwsdp 2.0 (Java Web Service Development Pack). With JDK 1.6 it is necessary to download a snapshot of the saaj library because of a bug in the released version (https://saaj.dev.java.net/servlets/ProjectDocumentList included in tws_Examples.zip). If you want to build the project with JDK 1.5 you need to regenerate the Web Service classes with wscomplile (chapter Step 2: Import the WSDL File).

## Create a New Project

Create a new project and import the WSDL file as described in Step 2: Import the WSDL File. Add the additional libraries to the project:




## Create Code to Call the Web Service

Add the following statement:

```
import java.io.*;
```

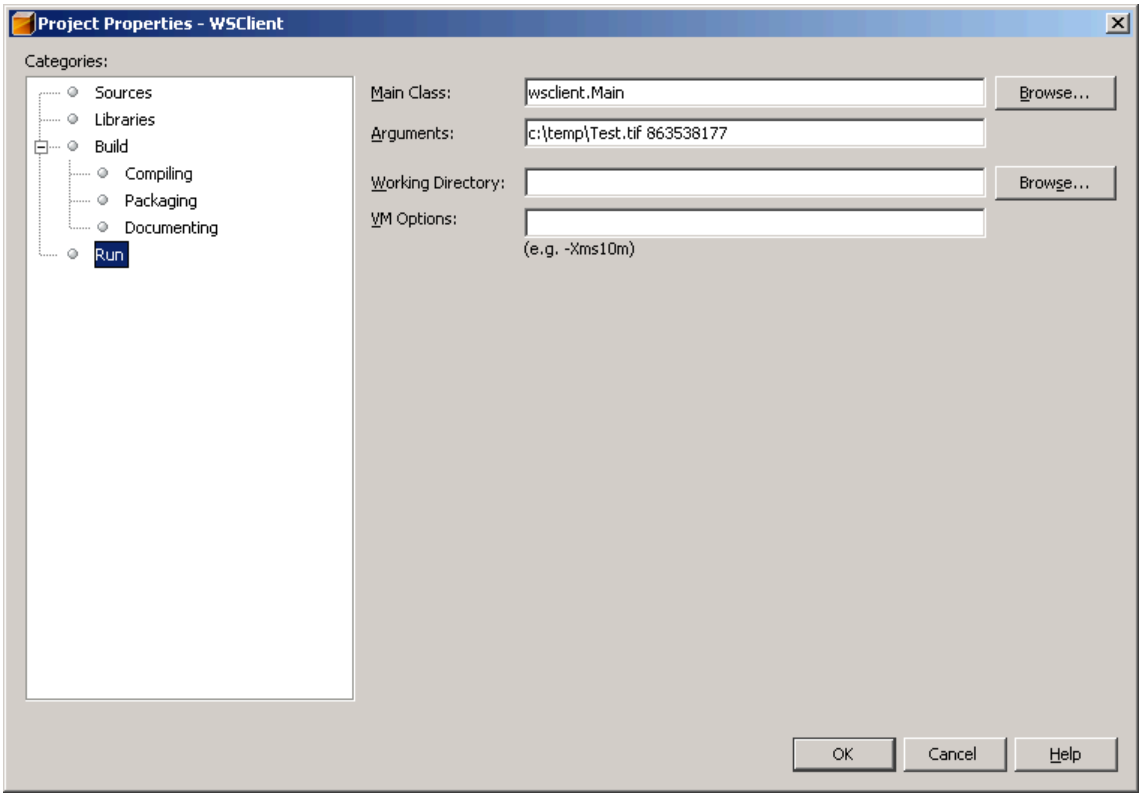
Fill the main function with the following commands:

```
String username = "guest";
String password = "";
String endpointaddress = "http://10.20.20.10/soap/fax";

try {
     File myfile = new File(args[0]);
     byte[] data = new byte[(int)myfile.length()];
     FileInputStream in = new FileInputStream(args[0]);
     int len;
     while ((len = in.read(data)) > 0) {
     }
     HeaderGen2 myheader = new HeaderGen2(args[1], "", "123456778", "Wolfi K",
                    "Testfax mit Java", false);
     BodyGen3   mybody = new BodyGen3(data);
     Stub stub = createProxy();

     stub._setProperty(javax.xml.rpc.Stub.USERNAME_PROPERTY, username);
     stub._setProperty(javax.xml.rpc.Stub.PASSWORD_PROPERTY, password);
     stub._setProperty(javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY, endpointaddress);


 System.out.println(stub._getProperty(javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY));
```

```
    FaxPortType myport = (FaxPortType) stub;
    myport.sendFax(myheader, mybody);
} catch (Exception ex) {
     ex.printStackTrace();
     System.out.println(ex.getMessage());
}
```

## Running the Example

1. Select **File** > **"WSClient" Properties** > **Run** and enter the filename and fax number:




2. Click **Run** to start the program.

# Example with PocketSOAP

PocketSOAP is Windows COM object that enables the use of web services.
You can download it from: http://www.pocketsoap.com. The current licensing for using PocketSOAP is Mozilla public license 1.1, see http://www.mozilla.org/MPL/MPL-1.1.html.

The advantage of using PocketSOAP (compared e.g. to Visual Studio .NET) is, that you must not install a complete development environment (Visual Studio .NET), you can start immediately using the TWS and e.g. VBScript, which is already shipped with the Windows operating system.

The disadvantage of using PocketSOAP is that PocketSOAP does generate a wrapper class based on the WSDL file. So you have to know the available methods and their parameters and you have no “Intellisense” support, as Visual Studio .NET provides.

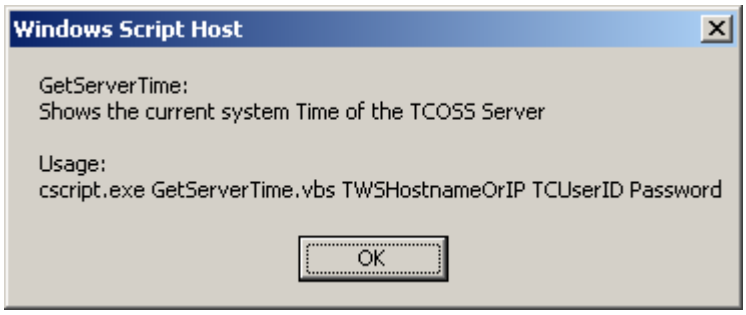
The following examples (all written using VB-Script) are available:

| Filename | Description |
|---|---|
| GetChannelList.vbs | Return a list of all TCOSS channels |
| GetServerTime.vbs | Return the current time of TCOSS server |
| OpenArchiveMsgPDF.vbs | Retrieves the last message of the logged in user from the TC Archive (PDF Format) |
| SendSimpleMessage.vbs | Sends a simple, text-based message via TWS |
| SetChannelStatus.vbs | Sets the status of one channel in TCOSS to ‘Waiting’ or ‘Continue’. |

If you start any script without command line parameters a dialog box with a short description of the usage and the required command line arguments are shown.

## **Example**: GetServerTime.vbs

Step 1: Start the script without parameters. You will get the following dialog box:




Step 2: Start the script from command line and specify the required arguments. An example command line is shown below:

C:\TOPCALL\tws\Examples\VBScript-PocketSoap>GetServerTime.vbs localhost tctech tctech

Step 3: If everything is ok, the program will respond with the TCOSS time as shown in the screen shot below:

# Examples with Perl

The example below shows a short perl script that logs in using TWS and returns the current time of the TCOSS server. The URL to TWS, the TCOSS user ID and the password must be given as command line parameters, e.g.

```
perl GetServerTime.pl localhost TCTECH tctech
```

```
#!perl -w
use SOAP::Lite;

# Check commandline parameters and assign them to variables
(@ARGV == 3) or die "\nUsage:\n$0 <TWSHostNameOrIP> <UserID> <Password>\n";

my ($hostname, $userID, $password) = @ARGV;

# Set HTTP basic authentication with the specified UserID/password
sub SOAP::Transport::HTTP::Client::get_basic_credentials {
...return "$userID" => "$password";
}

my $mySoapClient = SOAP::Lite
.........# define the SoapAction
                -> uri ('tcoss')
                # define the URL where the Webservice runs
                -> proxy ("http://$hostname/soap/tcoss")
                # define a handler subroutine, if a SoapFault or HTTP error occurs
on_fault (sub { my($soap, $res) = @_;
                     die "An Error occured: ", ref $res ? $res->faultstring :
$soap->transport->status, "\n"; })
                # this is important to have a HTTP Header SOAPAction filled with
                # only tcoss and not - as the default is - tcoss#set_get_time
on_action (sub{sprintf '%s', @_ })
                # uncomment to show the native SoapRequest/Response going over the
network
                #-> on_debug (sub{printf @_})
                ;
      print "\nCurrent TCOSS Time is: ", $mySoapClient->set_get_time()-
>valueof('//cl_time') , "\n";
```

All examples are designed to be used within a command prompt window. Following examples are currently available. Please check the code itself to see how to use specific functionality.

| Example | Parameters | Description |
|---|---|---|
| GetServerTime.pl | Hostname of TWS<br>TCOSS user ID<br>TCOSS password | Reads the current time from the TCOSS Server |
| SendSimpleMsg.pl | Hostname of TWS<br>TCOSS user ID<br>TCOSS password<br>KCS Service<br>KCS Number | Sends a predefined text (hard coded within the code) message to the specified Service and Number |
| GetChannelList.pl | Hostname of TWS<br>TCOSS user ID<br>TCOSS password | Displays the following information for each TCOSS channel: channel number, channel groups, current status (continue, waiting, Query, Client/Server) and channel description |

| Example | Parameters | Description |
|---|---|---|
| SetChannelStatus.pl | Hostname of TWS<br>TCOSS user ID<br>TCOSS password<br>Channel number<br>WAIT or CONT | Sets the status of the specified TCOSS channel to either “Waiting” or “Continue” |
| OpenArchiveMsgPDF.pl | Hostname of TWS<br>TCOSS user ID<br>TCOSS password<br>PDF Filename | Searches for the last message of the specified user ID stored in the Long term Archive (TC/ARCHIVE). Searching is done in the field “TS_REF” with the wildcard “*”. If a message is found, it is stored in PDF format to the filename specified as 4th command line parameter. |

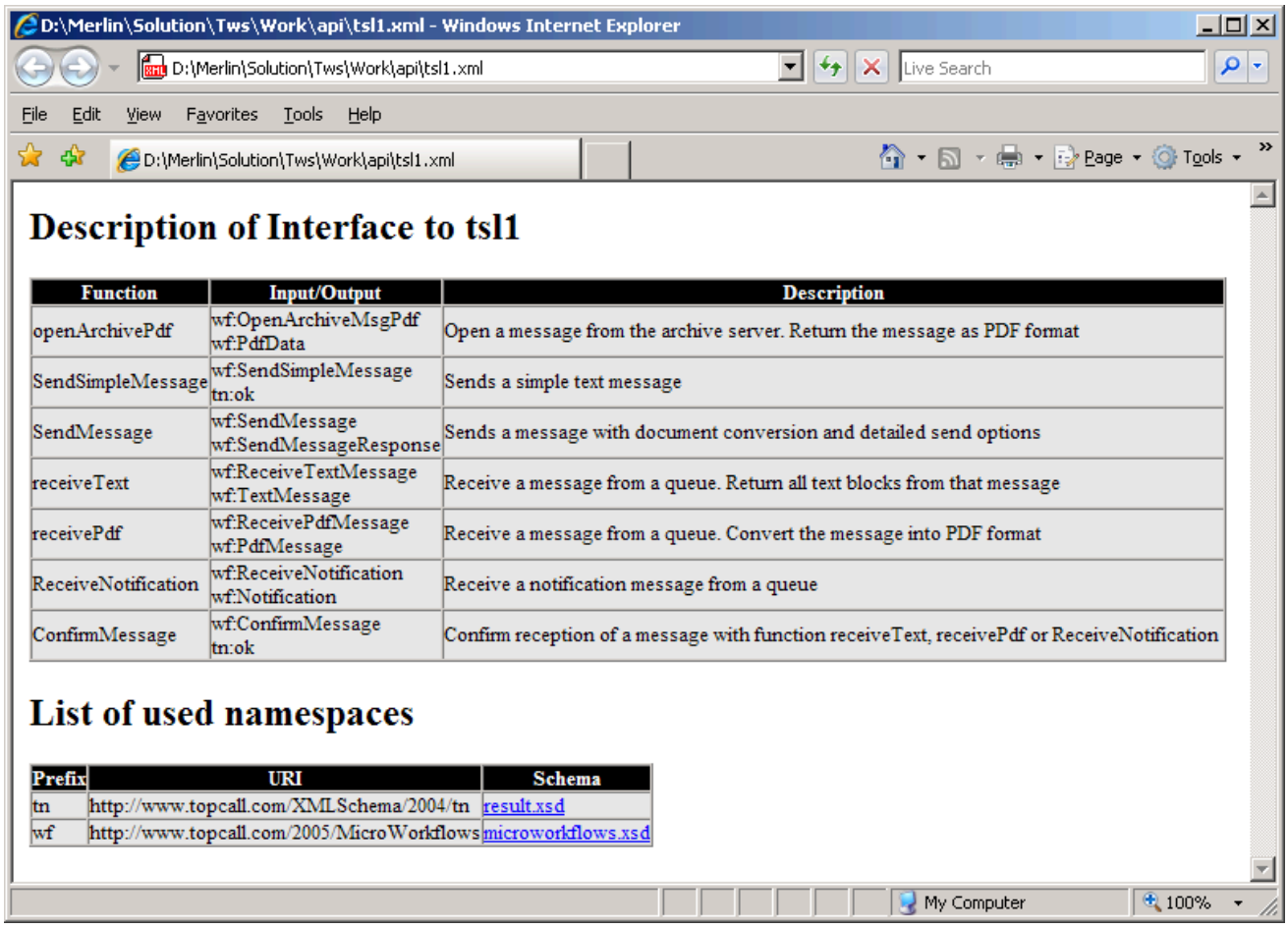
Chapter 7

# Web Service Functions

This chapter contains a list of all available web service functions. It is categorized by the WSDL files describing the functions.

You can get a short description of all functions for a specific WSDL file by opening the corresponding xml file in the API sub-directory with any tool that supports XML style sheets (e.g. Internet explorer).

**Example**: To get the interface for tsl1.wsdl open the file C:\topcall\tws\00\api\tsl1.xml with Internet Explorer. A sample screenshot is shown below:



The list of namespaces below contains the name and a link the schema files that defines the detailed parameter description for each call. It is recommended to use any XML tool (e.g. XML Spy) to view the schema files.

# Fax Interface Functions

The description of the web services interface between fax servers and applications can be found in a separate document, Web Services for Fax (*Web Services For Fax Technical Manual*). TWS makes use of the fax interface version 1.1.

# Micro Workflow Functions

These functions are defined in tsl.wsdl and tsl1.wsdl. The file tsl1.wsdl is a sub set of tsl.wsdl that does not contain functions dealing with TCXL formatted messages. The benefit of tsl1.wsdl is that it is very small compared to tsl.wsdl because it does not contain the schema for all data which may occur on the TCSI interface.

| Function | Input/Output | Description |
|---|---|---|
| GetInterfaceProperties | wf:GetInterfaceProperties<br>wf:InterfaceProperties | Get version and supported features of WS interface |
| openArchivePdf | wf:OpenArchiveMsgPdf<br>wf:PdfData | Open a message from the archive server. Return the message as PDF format |
| openArchiveXml 1) | wf:OpenArchiveMsg<br>tcsi:set_entry_archive | Open a message from the archive server. Return the message in TCXL format |
| SendSimpleMessage | wf:SendSimpleMessage<br>tn:ok | Sends a simple text message |
| SendMessage | wf:SendMessage<br>wf:SendMessageResponse | Sends a message with document conversion and detailed send options |
| ReceiveMessage | wf:ReceiveMessage<br>wf:ReceiveMessageResponse | Receive a message from a queue with image view and detailed content select options |
| receiveText | wf:ReceiveTextMessage<br>wf:TextMessage | Receive a message from a queue. Return all text blocks from that message |
| receivePdf | wf:ReceivePdfMessage<br>wf:PdfMessage | Receive a message from a queue. Convert the message into PDF format |
| ReceiveNotification | wf:ReceiveNotification<br>wf:Notification | Receive a notification message from a queue |
| ConfirmMessage | wf:ConfirmMessage<br>tn:ok | Confirm reception of a message with function receiveText, receivePdf or ReceiveNotification |
| ViewMessage | wf:ViewMessage<br>wf:ViewMessageResponse | Retrieve a message with image view and detailed content select options |
| TrackMessage | wf:TrackMessage<br>wf:TrackMessageResponse | Track a message on the KCS. Return a list of mail entries, one entry for each recipient of the message |

| Function | Input/Output | Description |
|---|---|---|
| GetMessageEntry | wf:GetMessageEntry<br>wf:MessageEntry | Return a mail entry showing the current status of a message on the KCS |
| CancelMessage | wf:CancelMessage<br>tn:ok | Cancel a message on the KCS |
| ReactivateMessage | wf:ReactivateMessage<br>tn:ok | Reactivate a message on the KCS |

List of used namespaces

| Prefix | URI | Schema |
|---|---|---|
| tn | http://www.topcall.com/XMLSchema/2004/tn | result.xsd |
| wf | http://www.topcall.com/2005/MicroWorkflows | microworkflows.xsd |
| tcsi | http://www.topcall.com/XMLSchema/2004/tcsi | tcsi.xsd |

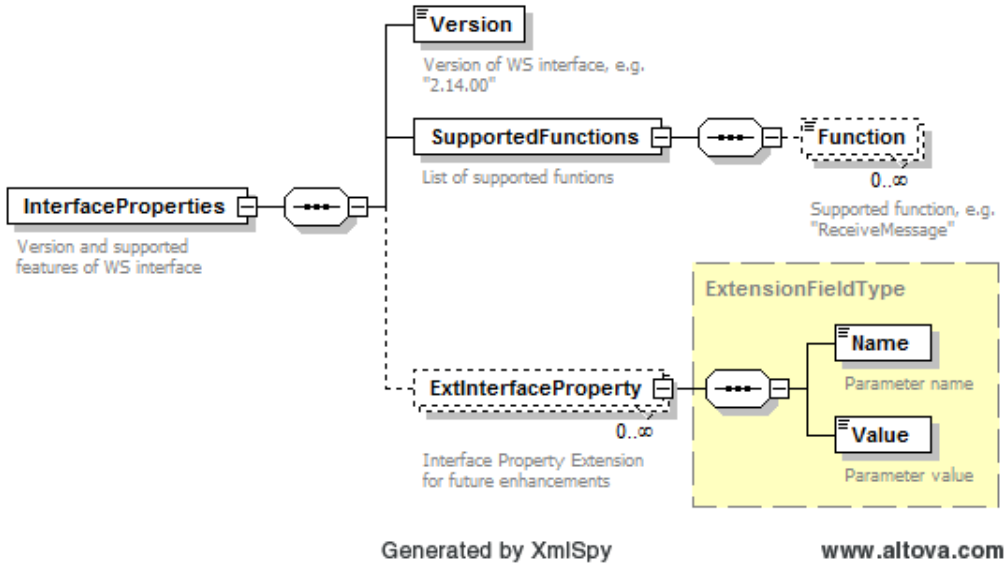
**Note 1)** Function openArchiveXml is available in tsl.wsdl only. All other functions can be found both in file tsl.wsdl and tsl1.wsdl.

## Get Interface Properties

The “GetInterfaceProperties” workflow returns information on the installed version and supported functions in that version. It has no input parameters.

Input:


Output Parameters:




**Example Output**:

```
<InterfaceProperties>
 <Version>2.16.04</Version>
 <SupportedFunctions>
  <Function>OpenArchiveMsg</Function>
  <Function>OpenArchiveMsgPdf</Function>
  <Function>SendSimpleMessage</Function>
  <Function>SendMessage</Function>
  <Function>ReceiveMessage</Function>
  <Function>ReceiveTextMessage</Function>
  <Function>ReceivePdfMessage</Function>
  <Function>ReceiveNotification</Function>
  <Function>ConfirmMessage</Function>
  <Function>ViewMessage</Function>
  <Function>TrackMessage</Function>
  <Function>GetMessageEntry</Function>
  <Function>CancelMessage</Function>
  <Function>ReactivateMessage</Function>
 </SupportedFunctions>
</InterfaceProperties>
```

The ExtInterfaceProperty entry with Name=”Type” indicates the used server (like the Type parameter in the GetFax interface) as described below:

| Value | Server |
|---|---|
| TWS | KCS with TWS |
| RWS | Connector to Rightfax server |
| BWS | Connector to BISCOM fax server |
| MC | KCIC-AEF Message Connector |

If this value does not exist TWS is assumed. Additional values are reserved for future use.

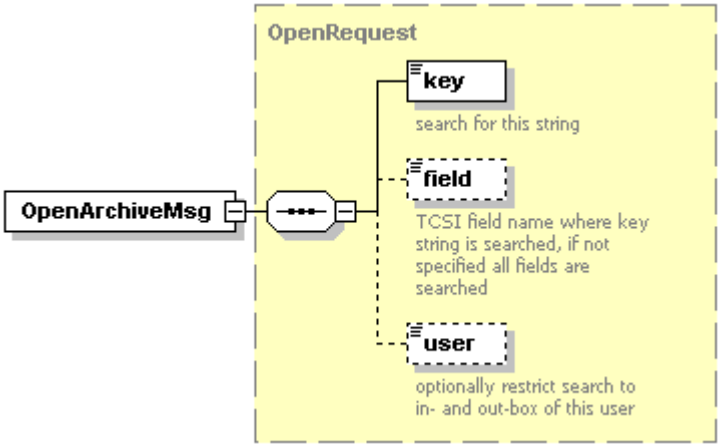
## Open Message from Archive Server

These functions are intended to retrieve a message from the archive server using a unique key. It is assumed that the messages originally sent via TCOSS already contained a unique key, e.g. some kind of reference number inside a custom header field. The unique message keys are kept by an application in an external database, on user request the message is fetched from the archive and displayed.

The microworkflow consists of the following steps: Open an archive search folder, take the first entry from the search result and open the message and (optionally) convert the message to PDF. If the search result is empty a fault is returned. There is no check whether the key is really unique, if the search result contains several matches the first (i.e., most recent) message is taken.

The two functions have identical input parameters. The openArchiveXml call returns the message in xml format with all the details of the complex TCSI message format. The openArchivePdf call returns the message in PDF format for easy viewing.

Input Parameters:




Output of openArchivePdf:


A single “PdfData” Element holding the PDF data in Base64 encoded form.

Output of openArchiveXml:

A “set_entry_archive” element with child elements as defined in the tcsi.xsd schema.

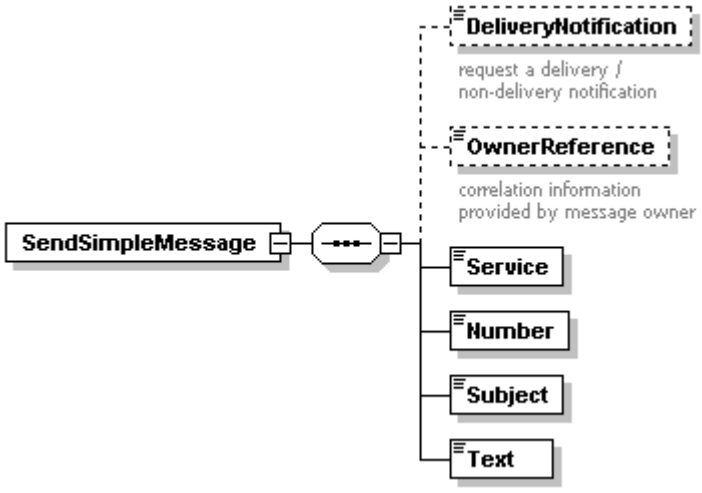
## Send Simple Text Message

The SendSimpleMessage allows you to post a text message to the TCOSS mail system for sending to a single recipient. A notification is requested by setting the optional input parameter “DeliveryNotification” to “POSITIVE”, “NEGATIVE” or “ALL” (default value is “NO”), correlation information to be returned in the notification may be passed in the “OwnerReference” field.

The workflow consists of these steps:

1. Get the current user from the TCOSS connector (configured or from basic authentication data)
2. Open the user's recipient entry in the TCOSS recipient store
3. Construct a message with the input data and the user's recipient entry as originator and post it to TCOSS

   Input Parameters:

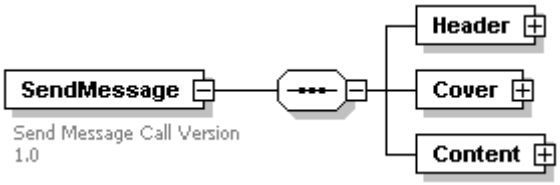



Output: An empty "ok" element if the message is posted successfully, a fault otherwise.

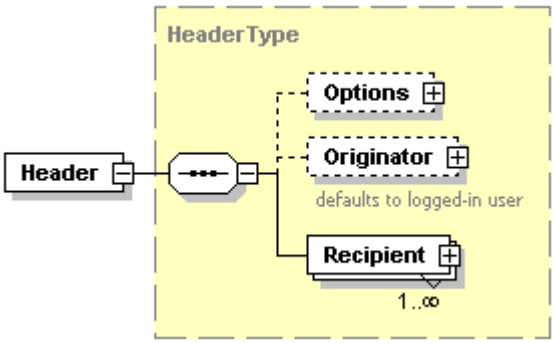
## Send Message

The SendMessage work flow puts a message into the Kofax Communication Server. It supports document conversion of the input message and some advanced send options like cover sheet and multiple recipients.

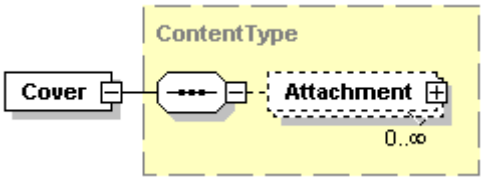
Input Parameters:

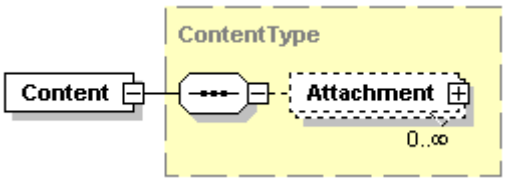



The input "Header" element holds send options, originator and recipients:




The “Cover” and “Content” elements hold a number of attachments:

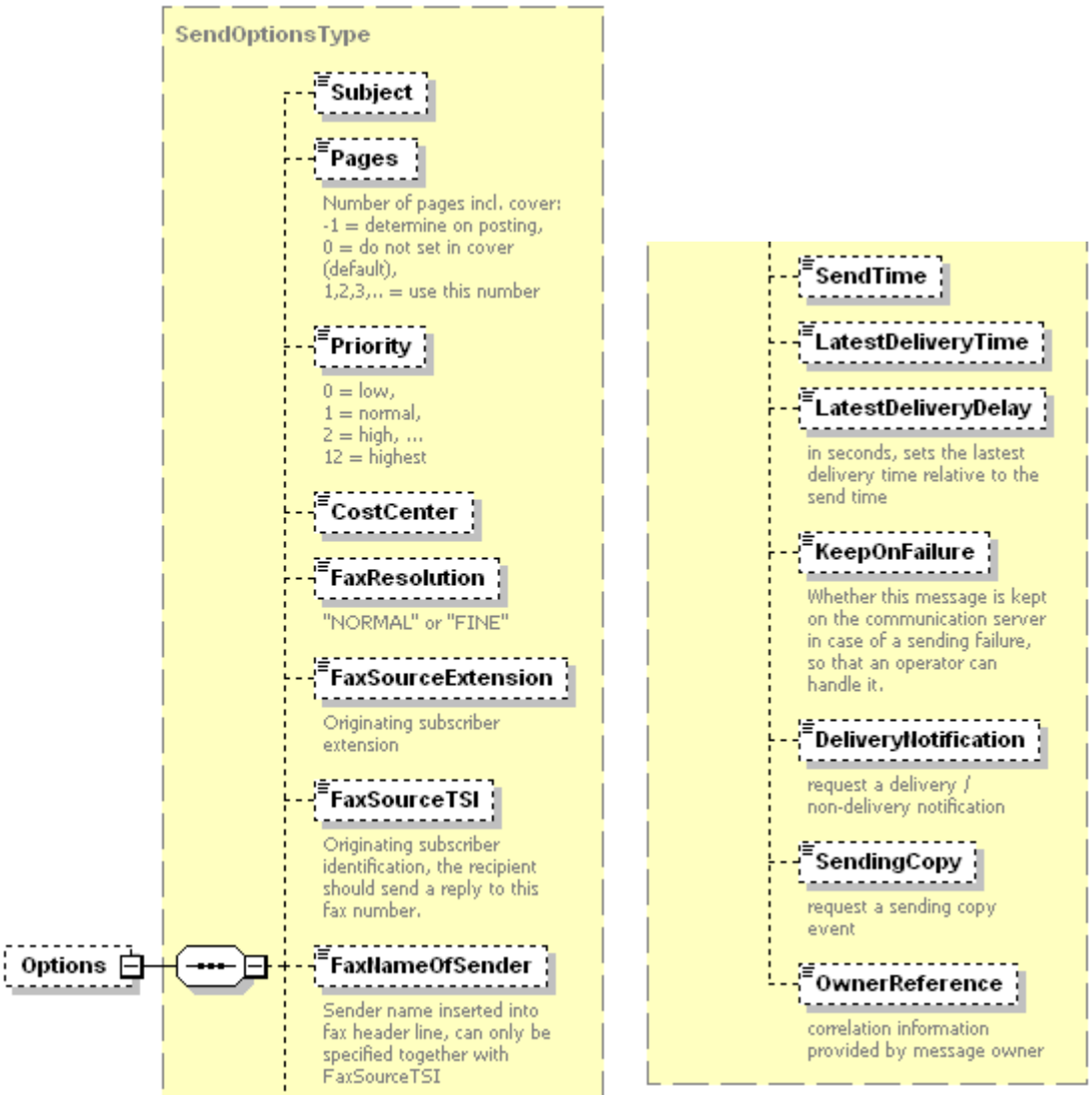


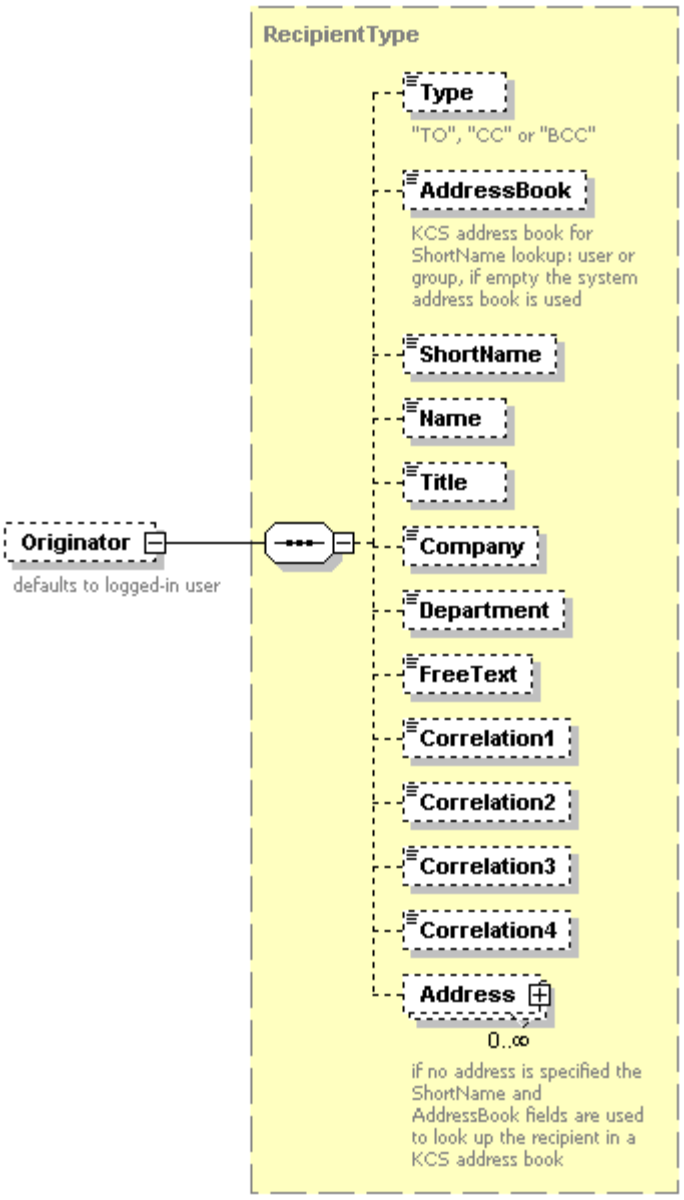




The send options allow to set a number of per-message send options. All elements are optional.

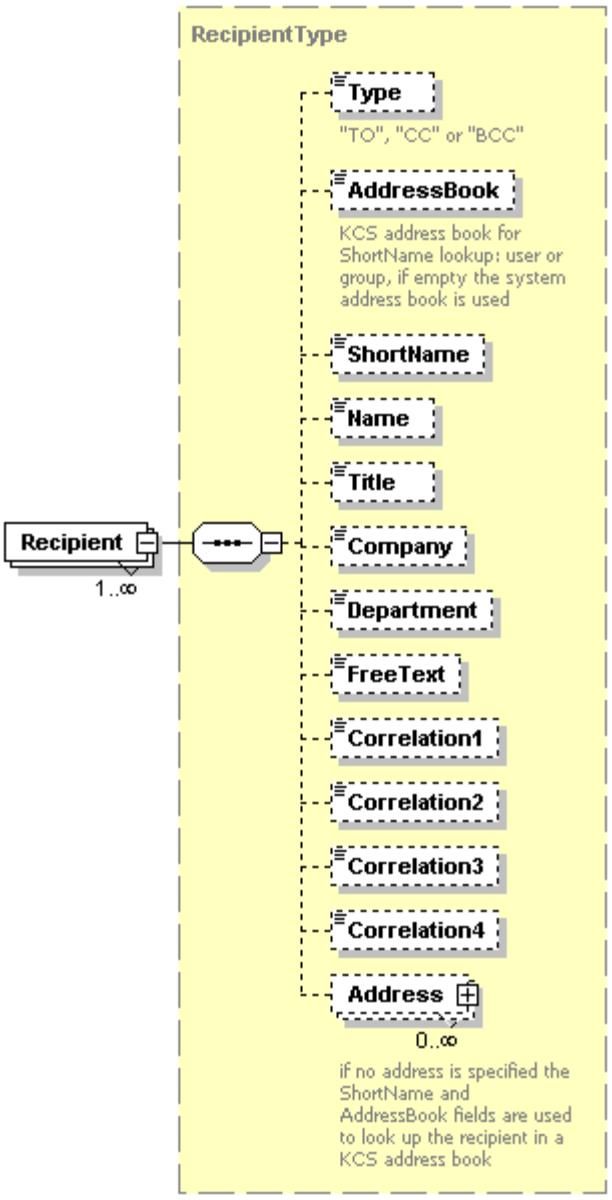
The Pages element contains the number of pages of a message including cover sheet. It can take the following values:

- –1 – The number of pages is calculated during posting of a message. This value affects performance.
- 0 – Do not set in cover (default).
- 1, 2, 3, … 999 – Use this number as number of pages


SendOptionsType
Options
Subject
Pages
Number of pages incl. cover:
-1 = determine on posting,
0 = do not set in cover (default),
1,2,3,.. = use this number
Priority
0 = low,
1 = normal,
2 = high, ...
12 = highest
CostCenter
FaxResolution
"NORMAL" or "FINE"
FaxSourceExtension
Originating subscriber extension
FaxSourceTSI
Originating subscriber identification, the recipient should send a reply to this fax number.
FaxNameOfSender
Sender name inserted into fax header line, can only be specified together with FaxSourceTSI
SendTime
LatestDeliveryTime
LatestDeliveryDelay
in seconds, sets the lastest delivery time relative to the send time
KeepOnFailure
Whether this message is kept on the communication server in case of a sending failure, so that an operator can handle it.
DeliveryNotification
request a delivery / non-delivery notification
SendingCopy
request a sending copy event
OwnerReference
correlation information provided by message owner




The message originator is optional, if it's not specified the recipient entry of the logged-in user is loaded from the KCS system address book. Originator and recipients share the same structure, and are processed in the same way, see recipient description below.

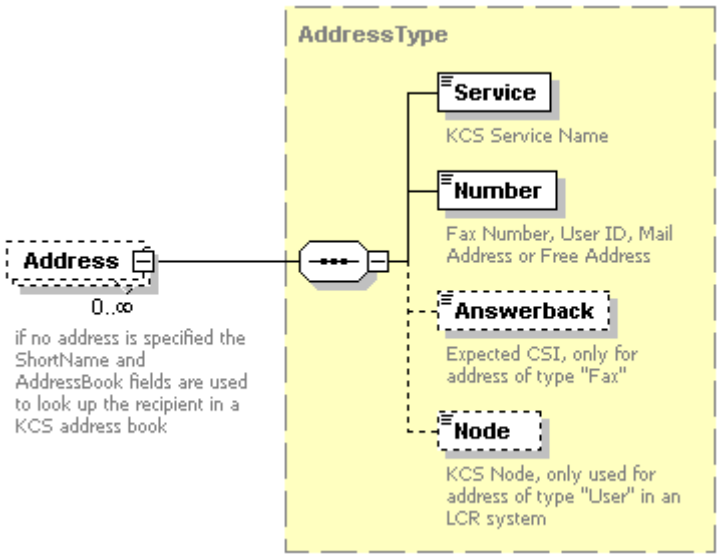



The recipient structure can be used in 2 alternative ways: as link to an entry in a KCS address book or as complete recipient specification.

If the “Address” element is missing the “ShortName” and optional “AddressBook” fields are used to look up the recipient in a KCS address book. The “Type” (“TO”, “CC” or “BCC”) is inserted if specified, the other fields like “Name”, “Title”, “Company” etc. are filled from the address book entry and not from the input structure.

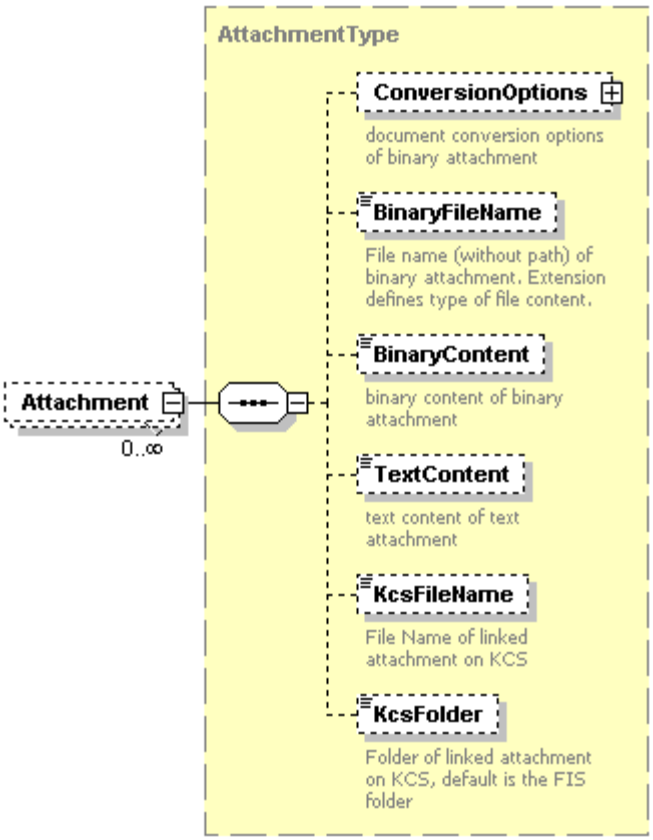
If at least one “Address” element is specified all fields from the input structure are mapped accordingly and the “Address” elements are transformed to the appropriate TCSI structure depending on how the

specified service is defined on the KCS. If more than one address is specified, the 2nd, 3rd etc. address is handled as alternative address.




The “Service” and “Number” fields of the address structure are mandatory. The optional “Answerback” field is only used with Fax and Telex addresses. The “Node” element is only relevant for KCS user IDs in a least cost routing system.

The message and cover content is defined as list of attachments:




The “Attachment” structure can be used in 4 different ways:

A) Binary content which has to be processed by the document converter

B) Transparent binary content

C) Text content

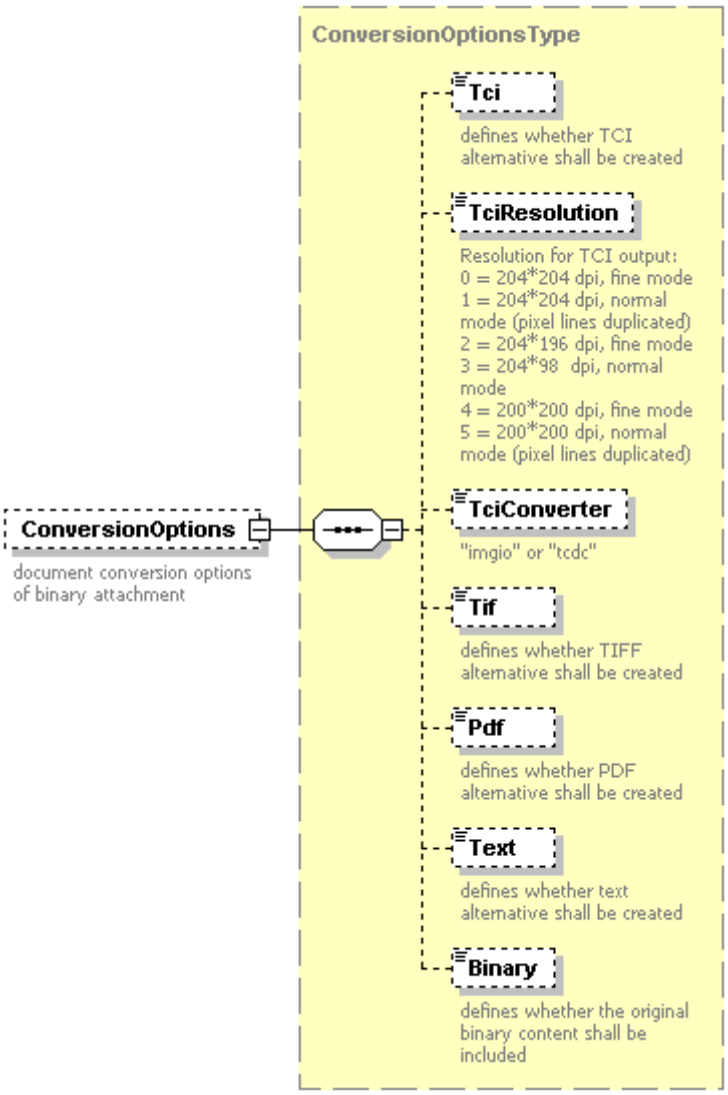
D) Linked attachment on KCS

In case A the “BinaryContent”, “BinaryFileName” and “ConversionOptions” elements are present.

In case B only the the “BinaryContent”, “BinaryFileName” elements are present.

In case C only the “TextContent” element is used.

In Case D the “KcsFileName” and optionally the “KcsFolder” elements are used.

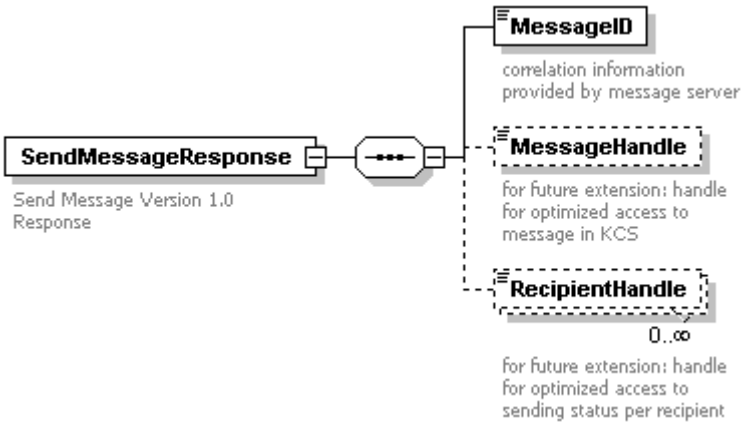



The “ConversionOptions” structure defines how the binary content is processed by the document converter.

Detailed parameters description:

| Name | Description | Default |
|---|---|---|
| TciResolution | Resolution for TCI output. Possible values:<br>0 … 204*204 dpi, fine mode<br>1 … 204*204 dpi, normal mode (pixel lines duplicated)<br>2 … 204*196 dpi, fine mode<br>3 … 204*98 dpi, normal mode<br>4 … 200*200 dpi, fine mode<br>5 … 200*200 dpi, normal mode (pixel lines duplicated)<br>Other values, or empty: printer defaults<br>For conversion via TCIMGIO, only 204*196 (for values 0,2,4) and 204*98 (for values 1,3,5) are used. | 0 |
| TciConverter | With this parameter, you can force conversion via TCIMGIO or TCDC. If this parameter is not present or empty, conversion via TCIMGIO is attempted first, and if it fails, conversion is attempted via TCDC.<br>imgio … force conversion via TCIMGIO<br>tcdc …. force conversion via TCDC | |

Output:




In the current TWS implementation only the “MessageID” element is returned.

## Receive Simple Message or Notification from TCOSS Queue

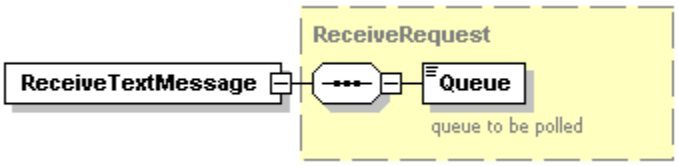
This group of four functions is designed to receive a text message, an image or a notification from a TCOSS queue (receiveText, receivePdf and ReceiveNotification calls) and to confirm the successful transfer of the message or notification (confirmMessage).

The message or notification returned by the receiveText, receivePdf or ReceiveNotification calls is taken out of the TCOSS queue with a timeout of one hour. If the message transfer is not confirmed to TCOSS within one hour the message will be put back into the queue and can be returned again by a receiveText, receivePdf or ReceiveNotification call. Several agents may receive from the same queue, e.g. for load balancing or fault tolerance.

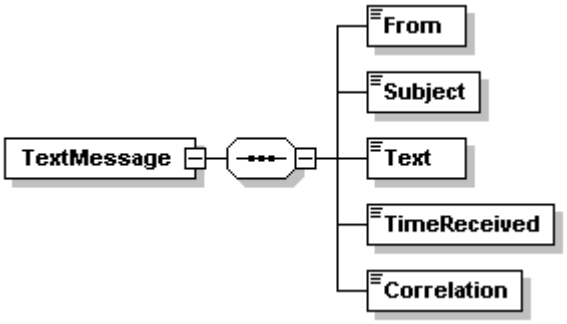
Mixed queues holding messages and notifications, or text and image messages, are not supported. Use only one of the receive functions (receiveText, receivePdf or ReceiveNotification) on a particular queue and make sure that the queue holds only messages of the appropriate type or only notifications.

Hint: The receiveText, ReceiveNotification and ConfirmMessage web service calls can be simulated and tested using the TWS web portal's "Test Server" window. Select for example the template "Workflow receive notification", enter the queue value in the "optional Parameters" text box after "Queue=" and click the button "Show formatted result".

Input of receiveText:

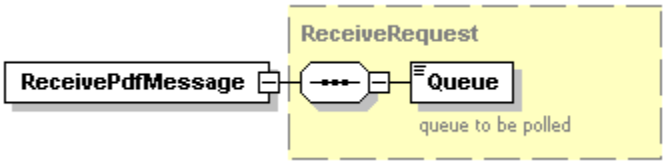



Output of receiveText:

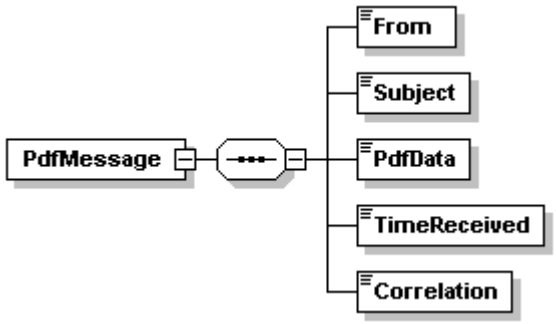



The "Text" element contains all text blocks from the TCOSS message. Image blocks and binary attachments are ignored.

Input of receivePdf:

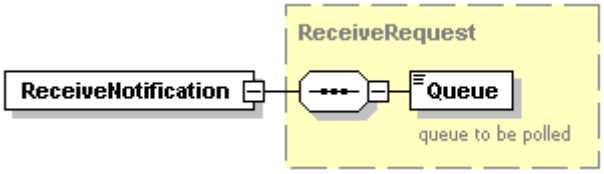



Output of receivePdf:



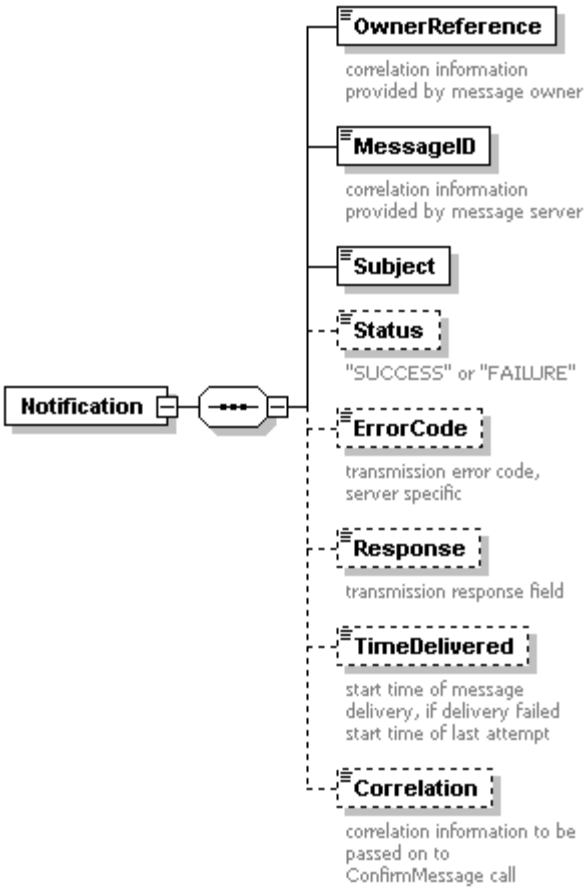

The “PdfData” Element holds the PDF data in Base64 encoded form. The PDF view includes text and image blocks of the original TCOSS message.

Input of ReceiveNotification:

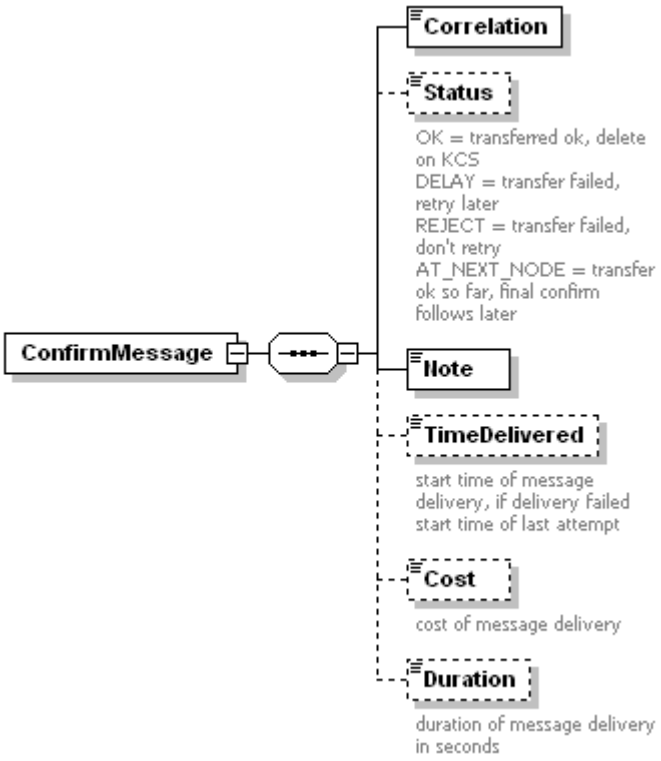



The polled queue should hold notifications only. Any regular message in that queue would also be taken out and converted into a “Notification” structure with all fields except “Correlation” missing.

Output of ReceiveNotification:




Input of confirmMessage:




The “Correlation” element is a Base64 coded string which should be taken unchanged from a previously received “TextMessage”, “PdfMessage”, “Notification” or “ReceiveMessageResponse” structure.

The “Status” element selects how the message is handled further on the KCS. Use “OK” to confirm a transferred message.

Status “DELAY” puts the message back into the KCS queue for retrieval at a later time. This provides a limited number of retries, if that number is passed it will work out like the reject option, to avoid an endless loop.

Status “AT_NEXT_NODE” stops the confirm timeout supervision. The message waits without timeout for a final confirm message call with status “OK”, “DELAY” or “REJECT”.

The “Note” is optional and sets a descriptive string in the TCOSS delivery information, e.g. “transferred to system xxxx”. The “Note” is displayed within the TCfW column “Response”. The optional “TimeDelivered”, “Cost” and “Duration” fields may be used to set specific delivery information.

**Note** The “Note”, “TimeDelivered”, “Cost” and “Duration” fields are stored by TCOSS only if the queue where the message has been taken from (see receive functions above) is defined with the ”Visible in Outbox” flag checked in the general section of the TCOSS user profile.

**Output**:

An empty “ok” element if the message is confirmed successfully (i.e. the correlation was correct and the message was still open in TCOSS), a fault otherwise. If the message to be confirmed does not exist

anymore or it has already been confirmed, a fault with error category ErrLogicState is returned. In this case it does not make sense to retry the ConfirmMessage call by the client.

## Receive Message from TCOSS Queue

The ReceiveMessage call is a more general function to receive a message from a TCOSS queue, compared to the simple receiveText and receivePdf functions described in the previous chapter.

The application sends a “ReceiveMessage” request including options to select what parts of the message are returned and in what format. KCS returns the message in the requested format. The application confirms the transfer of the message using a “ConfirmMessage” web service call.

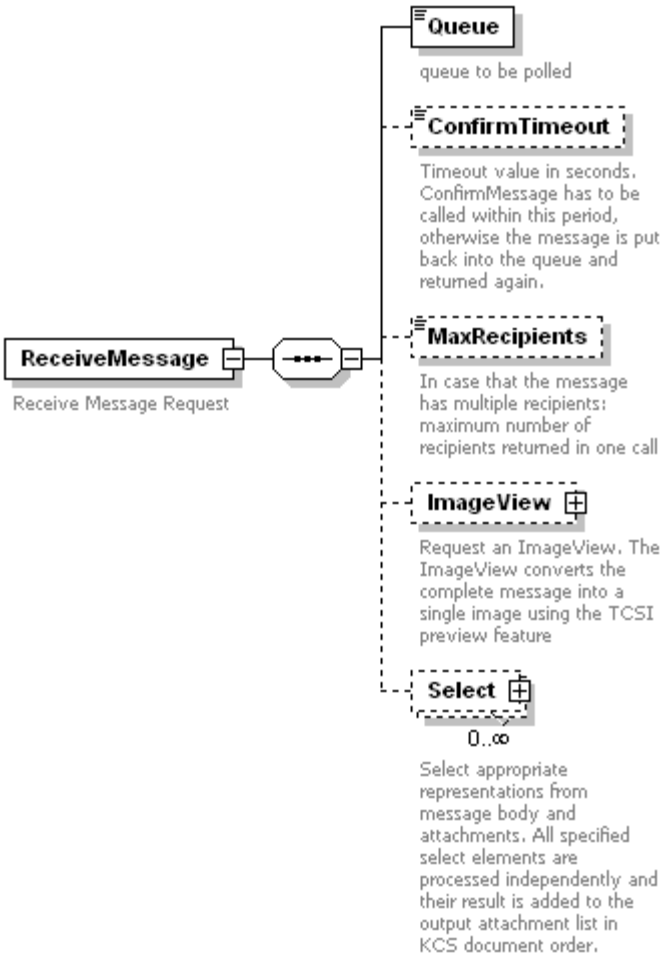
If the KCS queue is empty the response to the “ReceiveMessage” request is an empty shell without message content. No “ConfirmMessage” call is issued in this case.

If the KCS queue holds multiple entries for the same message (to different recipients), the response to a “ReceiveMessage” request may consist of a message and a list of recipients (each requiring a “ConfirmMessage” call to confirm). The application can specify the maximum number of entries it wants to have returned using an option in the “ReceiveMessage” request. This maximum number can be set to 1 so that only a single entry is returned.

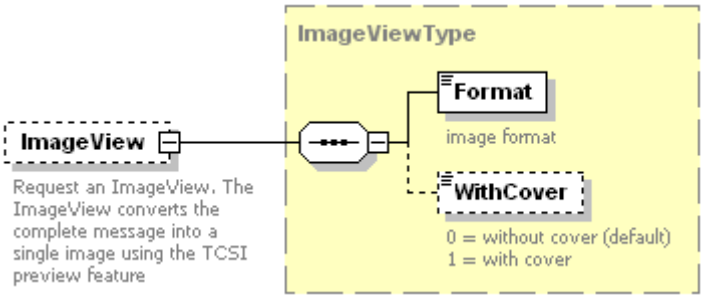
If an application gets a message in response to a “ReceiveMessage” request, but fails to confirm its transfer within a timeout period, the message is put back into the KCS queue to be returned again as response to another “ReceiveMessage” request at a later time.

Options in the “ReceiveMessage” request allow the application to get the message as a single image in TIFF or PDF format. Additionally / alternatively the application can select content parts of the message depending on type (text, image, binary content), rank (original / alternative) and extension (file extension, only for binary content), these parts to be returned in the response. It is also possible to request a response holding only the message entry without any message content, and to retrieve the message content later using the “ViewMessage” function.

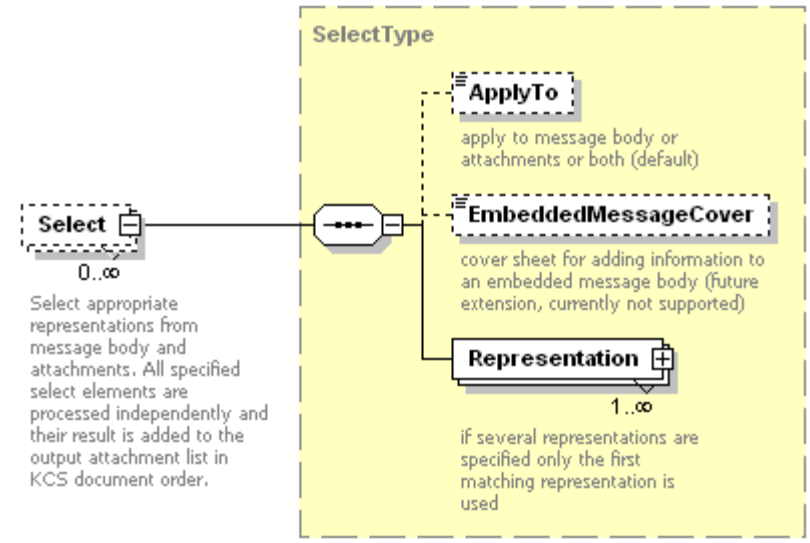
Input Parameters:

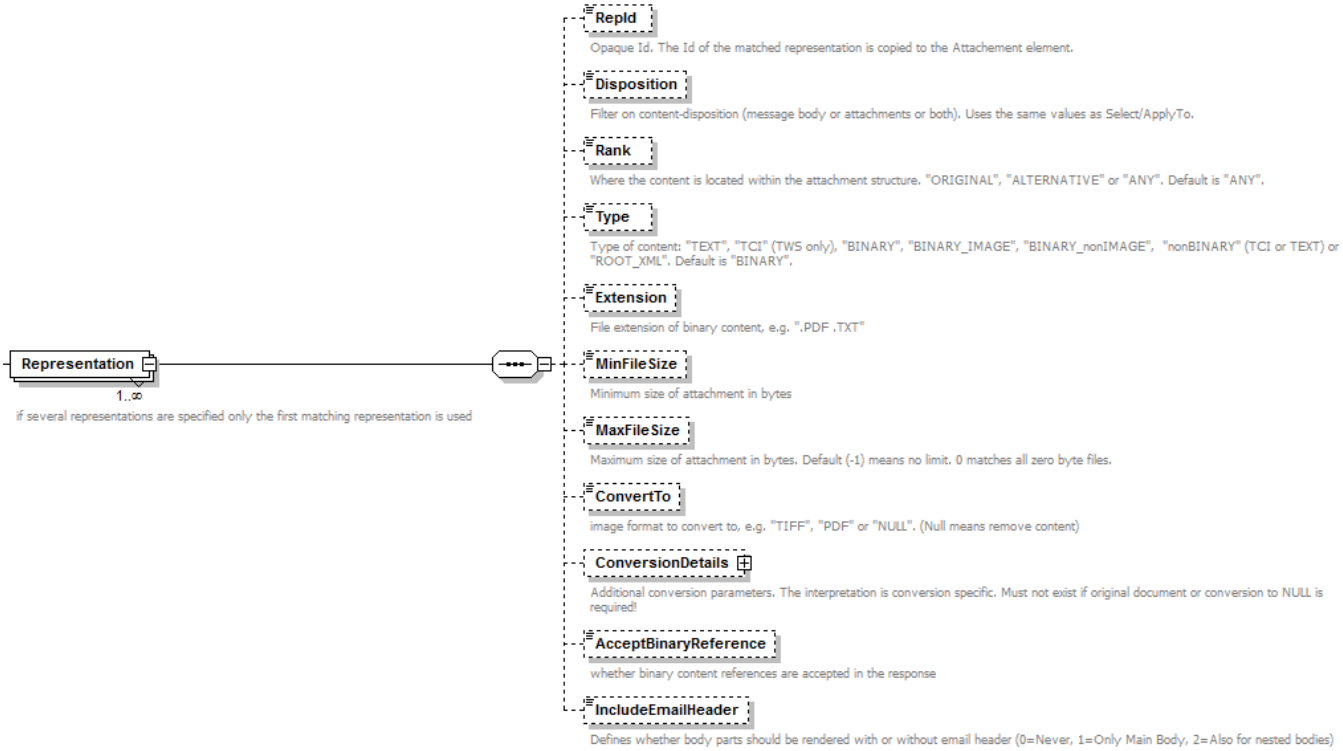



Optional ImageView request:



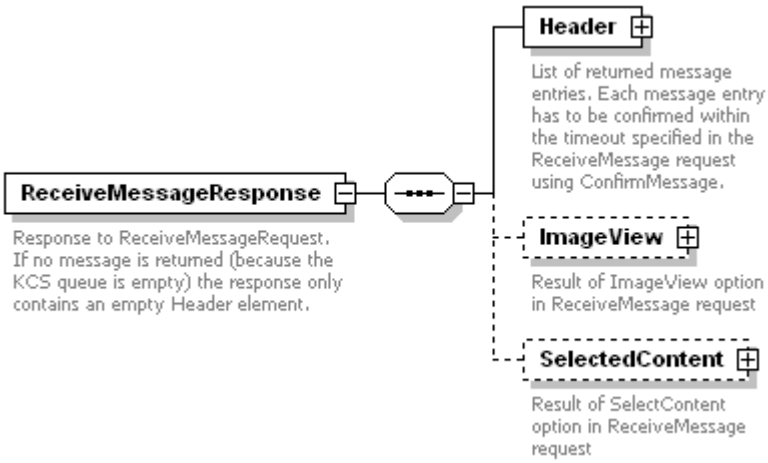

Optional Select list:



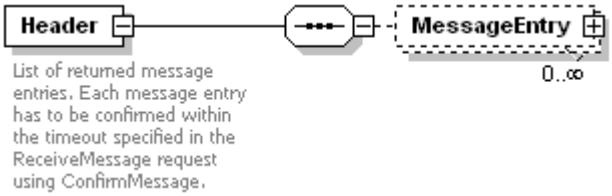




The “Representation” identifies message content by “Rank” and “Type”. The “Extension” is an additional filter for content if type “BINARY” only. The optional “ConvertTo” element may be used to convert the selected content into a TIFF or PDF image. Content of Type “TCI” can only be returned converted into TIFF or PDF.

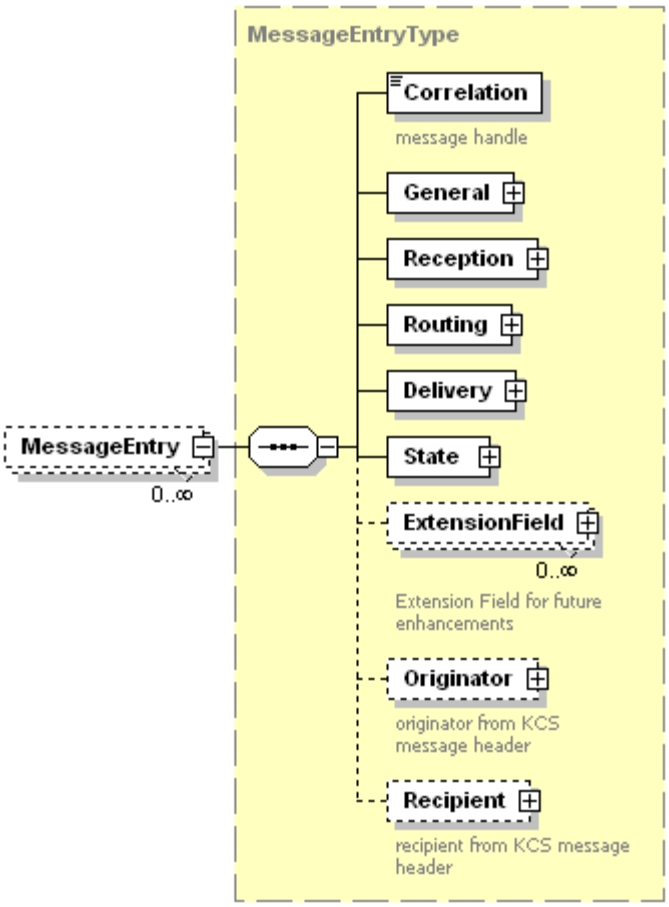
Output Parameters:



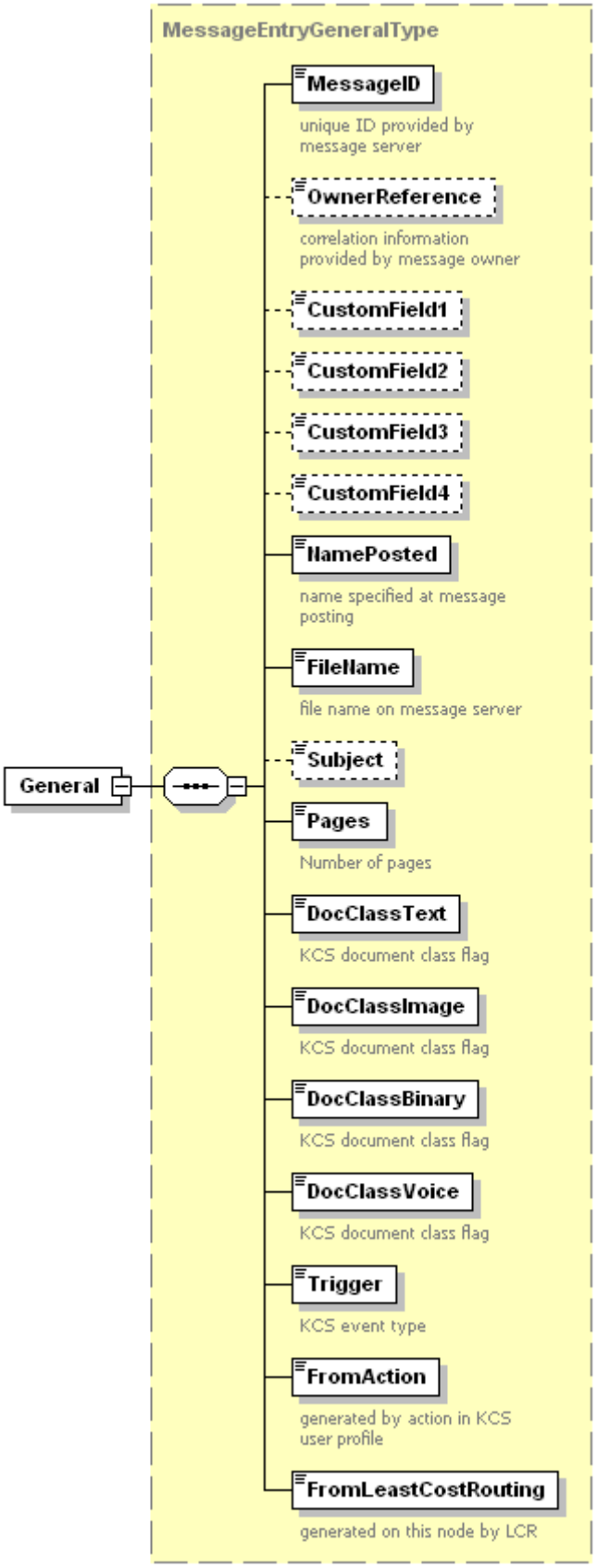

The Header holds a list of message entries:

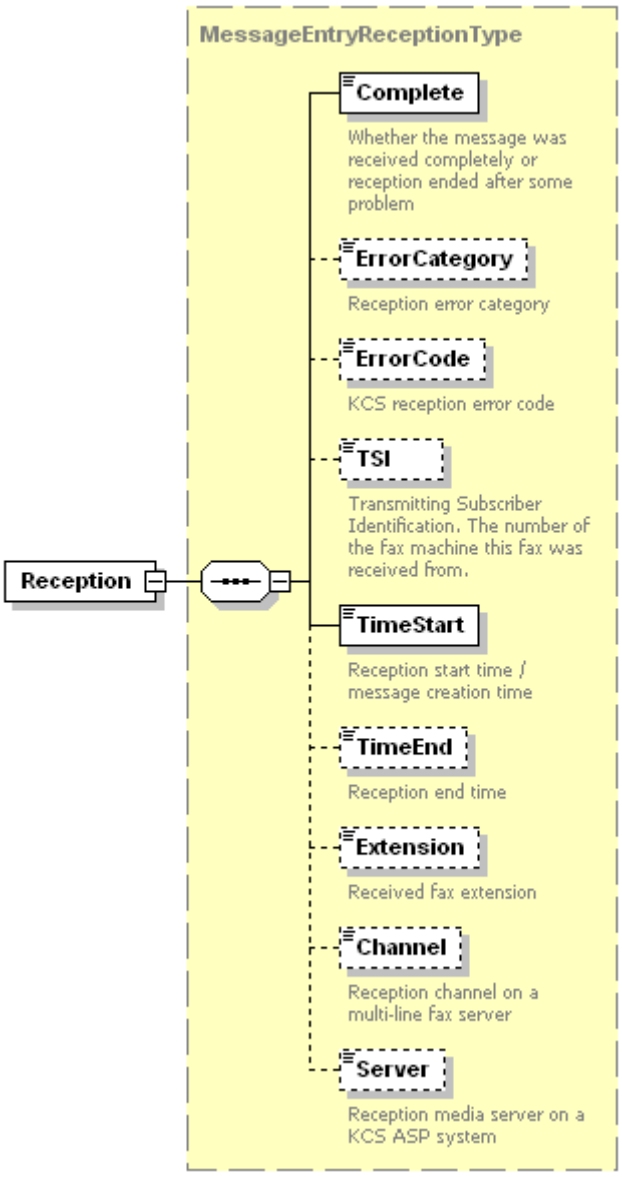



Each message entry is structured like this:

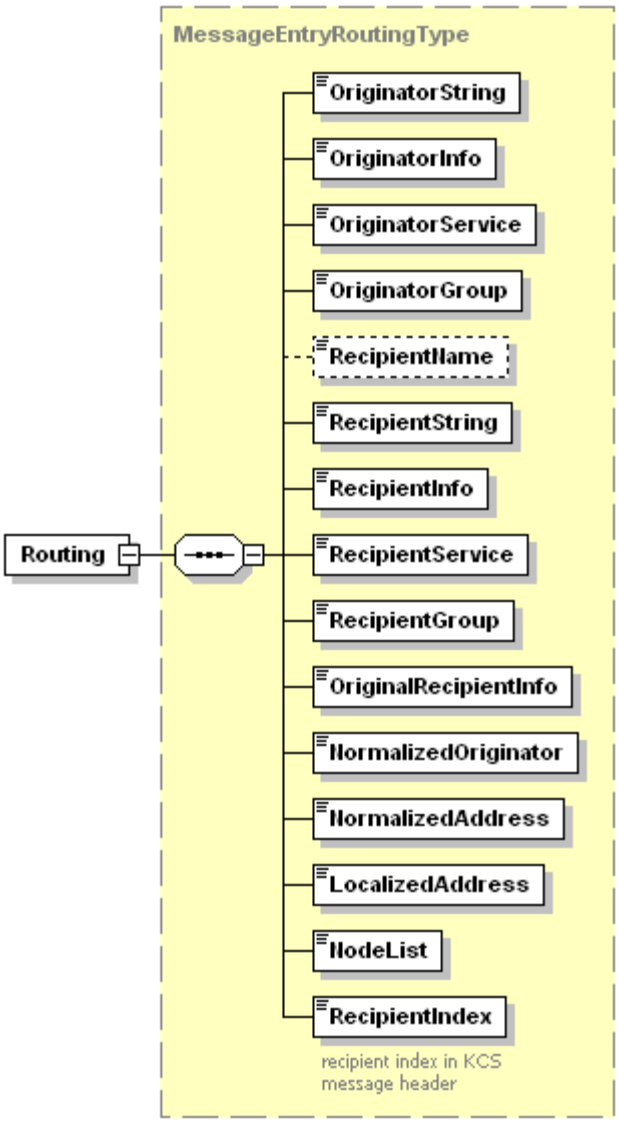



The Correlation element from the message entry serves as input parameter to the “ViewMessage” and “ConfirmMessage” functions. The “General” section holds information not related to specific recipients:


MessageEntryGeneralType
General
MessageID
unique ID provided by message server
OwnerReference
correlation information provided by message owner
CustomField1
CustomField2
CustomField3
CustomField4
NamePosted
name specified at message posting
FileName
file name on message server
Subject
Pages
Number of pages
DocClassText
KCS document class flag
DocClassImage
KCS document class flag
DocClassBinary
KCS document class flag
DocClassVoice
KCS document class flag
Trigger
KCS event type
FromAction
generated by action in KCS user profile
FromLeastCostRouting
generated on this node by LCR

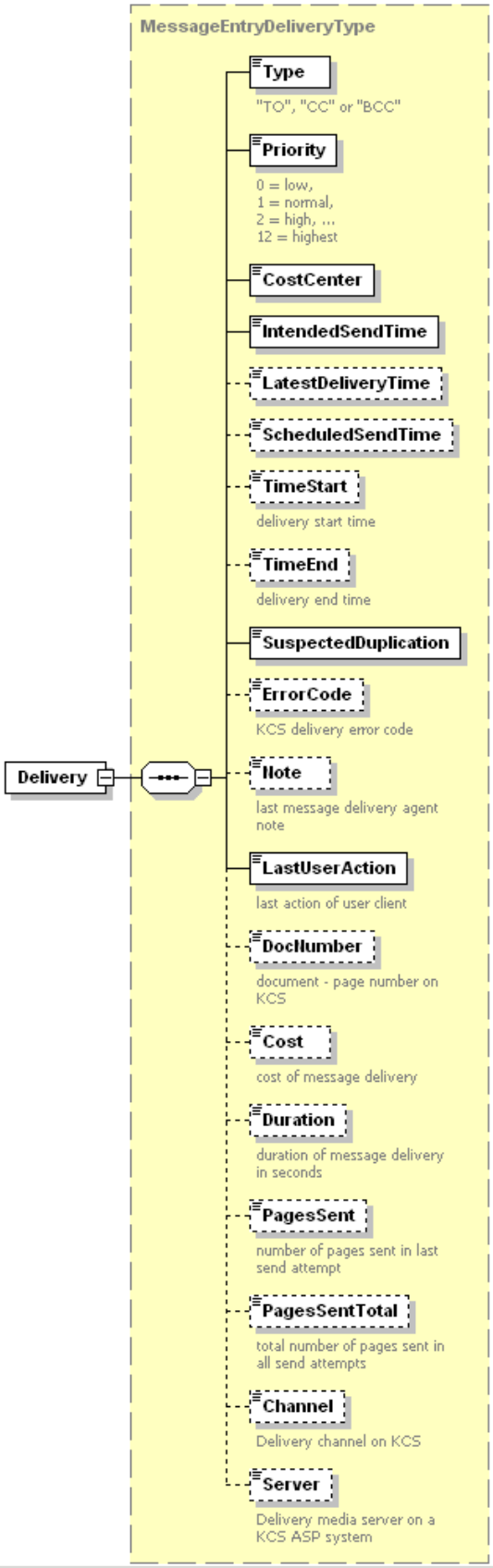



The “Reception” section is useful it it’s a received fax message and can be ignored otherwise.

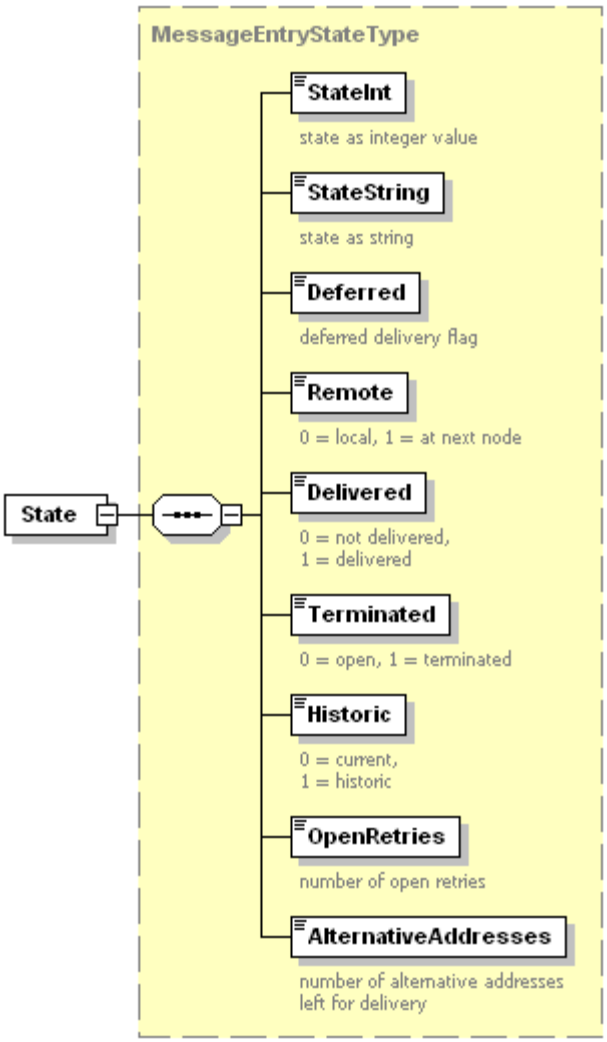



The Routing section gives recipient and originator information.

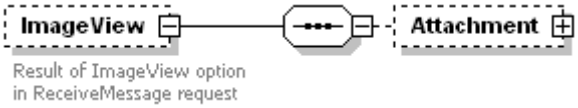
The Delivery section holds information related to message delivery:

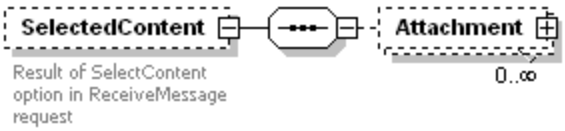

MessageEntryDeliveryType
Delivery
Type
"TO", "CC" or "BCC"
Priority
0 = low,
1 = normal,
2 = high, ...
12 = highest
CostCenter
IntendedSendTime
LatestDeliveryTime
ScheduledSendTime
TimeStart
delivery start time
TimeEnd
delivery end time
SuspectedDuplication
ErrorCode
KCS delivery error code
Note
last message delivery agent note
LastUserAction
last action of user client
DocNumber
document - page number on KCS
Cost
cost of message delivery
Duration
duration of message delivery in seconds
PagesSent
number of pages sent in last send attempt
PagesSentTotal
total number of pages sent in all send attempts
Channel
Delivery channel on KCS
Server
Delivery media server on a KCS ASP system

The State section details the message's state:




The result of the ImageView options is a list of attachments, a similar structure holds the result of the "Select" request list:




RecMessageAttachmentType
Attachment of received message
RepId
Id from matched representation
HierarchicalPosition
hierarchical position within message on KCS, e.g. "1_2_0"
Rank
Where the content is located within the attachment structure on KCS, "ORIGINAL" or "ALTERNATIVE".
Type
Actual type of content: "TEXT", "TCI" (TWS only), "BINARY_IMAGE", "BINARY_nonIMAGE" or "ROOT_XML"
Application
TS_APPL_ID field
Comment
TS_COMMENT field
ExtractedFromFile
Filename from where it is extracted
ContentID
MIME content-id field
ContentDisposition
MIME content-disposition
CreationDate
MIME creation-date property (of original attachment)
ModificationDate
MIME modification-date (of original attachment)
Size
MIME size property (of original attachment)
LongFileName
TS_LONG_FILE_NAME field. Extension may not match type of binary content (this is the original file name)
ConvertTo
requested conversion: "TIFF" or "PDF" or "" (empty string means no conversion requested)
faultstring
Signals a failure in document conversion, image view, open select or xml parsing, e.g. "document conversion failure" + fault string from document converter. Is also used to signal an unzip error: "decompression failure" + fault string.
BinaryFileName
File name (without path) of binary attachment. Extension defines type of binary content.
ContentType
MIME type of content (not provided in case of document conversion)
ExtensionField
0..∞
old style extension field for enhancements, deprecated
Extensions
Extensions for future enhancements
BinaryContent
binary content of binary attachment
BinaryReference
Reference to binary content, as an alternative to the BinaryContent element
TextContent
text content of text attachment
StructuredContent
XML content of attachment selected by Representation of Type "ROOT_XML". (Other or invalid XML is returned as BinaryContent)
Generated by XMLSpy
www.altova.com

The Attachment structure contains information where in the original KCS message it is coming from (“HierarchicalPosition”, “Rank”). Only one of the “BinaryContent” and “TextContent” elements is actually used, depending on the type of the attachment.

## Receive Message Error Handling

This section describes the error handling for ReceiveMessage function.

### Total Failure

In case of total failure the function returns a SOAP Fault element.

Returned xml on total failure (example):

```
<s:Fault xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <faultcode>s:Server</faultcode>
  <faultstring>612 server temporarily busy</faultstring>
  <detail>
    <t:result xmlns:t="http://www.topcall.com/XMLSchema/2004/tn">
      <t:code>558891526</t:code>
      <t:info>612 server temporarily busy</t:info>
    </t:result>
  </detail>
</s:Fault>
```

### Message queue empty

If the “ReceiveMessage” function fails to get a message because the polled queue is empty it returns a “ReceiveMessageResponse” containing an empty “Header” element:

```
<wf:ReceiveMessageResponse xmlns:wf="http://www.topcall.com/2005/MicroWorkflows">
 <wf:Header/>
</wf:ReceiveMessageResponse>
```

### Partial failure

If the “ReceiveMessage” function fails partially, e.g. in converting one attachment, it will return a regular response containing a “MessageEntry” element and any correctly retrieved and converted attachments within the “ImageView” and/or “SelectedContent” parent elements.

To report the failure the response will also include one or more “Attachment” elements without “BinaryContent” or “TextContent” holding an “ExtensionField” with Name “faultstring”. The “Value” element contains a string describing the problem.

Example of "Attachment" reporting failure:

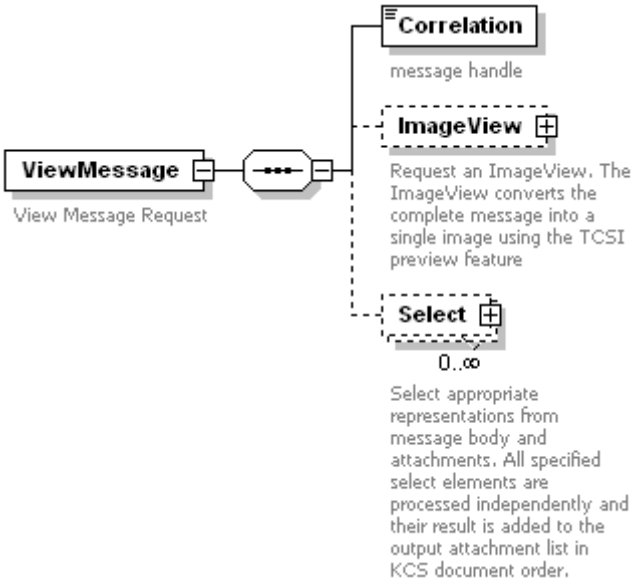
```
<wf:Attachment xmlns:wf="http://www.topcall.com/2005/MicroWorkflows">
<wf:HierarchicalPosition>2</wf:HierarchicalPosition>
<wf:BinaryFileName>IMAGEIO.PDF</wf:BinaryFileName>
 <wf:ExtensionField>
  <wf:Name>faultstring</wf:Name>
  <wf:Value>document conversion failure</wf:Value>
 </wf:ExtensionField>
</wf:Attachment>
```

## View Message

The “ViewMessage” retrieves the content of a message on the KCS. The presentation options are the same as for the “ReceiveMessage” workflow, the response structure is similar to the “ReceiveMessage” response.

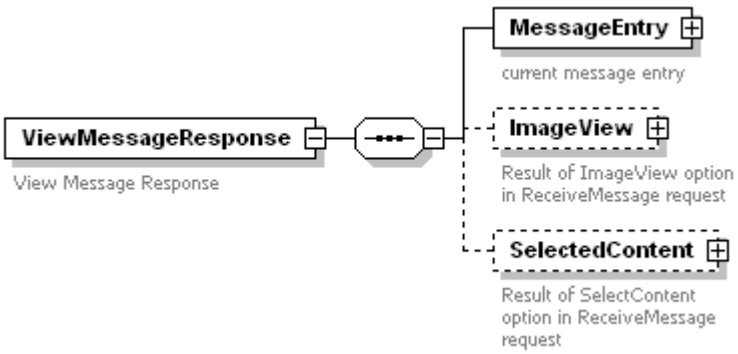
The main difference between “ViewMessage” and “ReceiveMessage” is that view does not transfer ownership of the message to the application, the status of the message on KCS is not changed by viewing it. Consequently the view message call is not followed by a confirm message call.

Input Parameters:




The “Correlation” element can be copied from a previous receive message, track message or a get message entry response.

Output Parameters:



See detailed description of response elements in “Receive Message from TCOOS queue” chapter. The error handling is the same as for the “ReceiveMessage” function, except that the “empty queue” error case does not exist.

## Track Message on KCS

The “TrackMessage” function allows to track a message on the KCS.

The request contains the “MessageID” field returned by the “SendMessage” web service call. The message ID may also be taken from an inbox or outbox view (field “ts_tc_msg_id”).

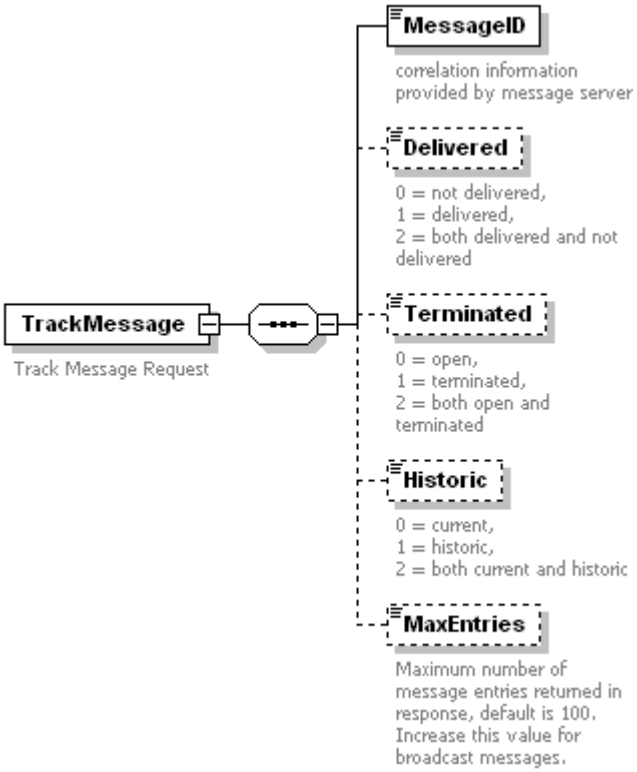
The response is a list of mail entries, one entry for each recipient of the message. The mail entries show to which recipients the message was delivered and to which recipients delivery failed or is still in progress.

Each mail entry contains a handle to be used for further operations like cancel or reactivate, handles from finished entries however are not suitable for these operations.

Restriction:

The maximum number of mail entries configured on the KCS has to be sufficiently large, considering the total message throughput of the KCS and the amount of time passed between the original “SendMessage” call and the message tracking call, so that all historic mail entries of the original send operation are still available (and have not been re-used and overwritten). Otherwise some or all historic mail entries are missing in the response.

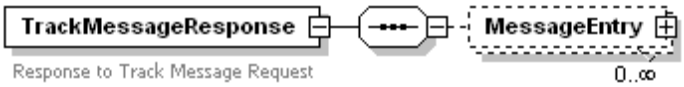
Input Parameters:



The “Delivered”, “Terminated” and “Historic” elements allow to restrict the extracted list of mail entries depending on their state. Default value for all three is “2” which means no restriction.

The “MaxEntries” option limits the number of entries returned in the response. The default of 100 prevents response lists with thousands of entries being generated if the MessageID was specified as “*”.

Output Parameters:

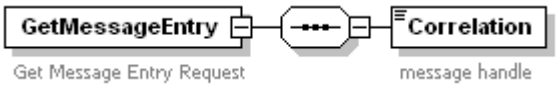



See a detailed description of the “MessageEntry” structure in the previous chapter.

## Get Message Entry

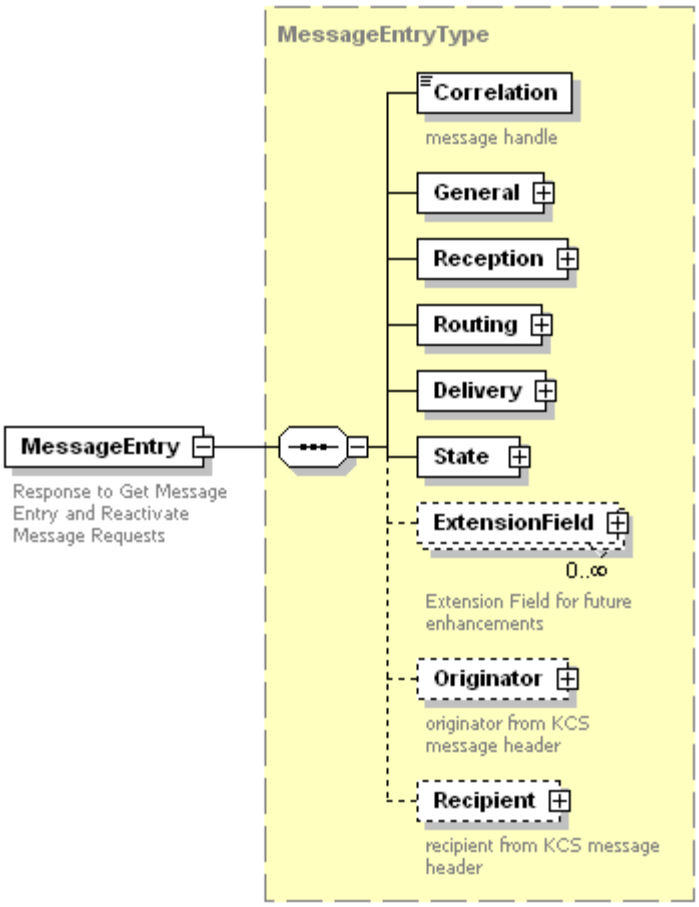
The “GetMessageEntry” workflow retrieves the current status of a message on the KCS. It returns a single message entry.

Input Parameters:




The “Correlation” element can be copied from a previous receive message, track message or a get message entry response.

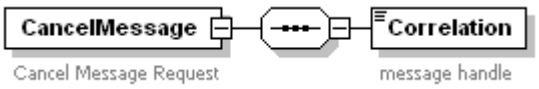
Output Parameters:




## Cancel Message

The “CancelMessage” workflow allows to cancel a message on the KCS.

Input Parameters:




The “Correlation” element can be copied from a previous track message or a get message entry response.

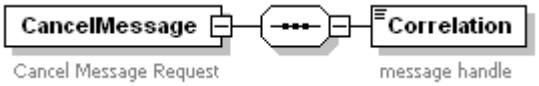
Output: An empty “ok” element if the message is cancelled successfully, a fault otherwise.

## Reactivate Message

The “ReactivateMessage” workflow allows to reactivate a message on the KCS. Reactivation means that the message gets the state of a newly posted message, with retry counters reset and the recipient

rerouted so that the first alternative address is tried first. The latest delivery timeout, if expired, is reset to the configured default value.

Input Parameters:




The “Correlation” element can be copied from a previous track message or a get message entry response.

Output: An empty “ok” element if the message is reactivated successfully, a fault otherwise.

# TCOSS Functions

These functions are defined in file tcoss.wsdl. The table below lists all available functions with a short description. More details about these functions can be found in the TCSI manual.

| Function | Input/Output | Description |
|---|---|---|
| tcsiGetLicenseState | tcsi:set_get_license_state<br>tcsi:set_state_license_store | Returns number of licenses, old / new licensing mode, end of test period |
| tcsiSetLicenseState | tcsi:set_state_license_store<br>tn:ok | Switch to new licensing mode, switch to test mode |
| tcsiGetRegistrationState | tcsi:set_get_registr_state<br>tcsi:set_state_n_max_reg | Get state of registration store (current / maximum number of entries) |
| tcsiGetSystemResources | tcsi:set_get_sys_resources<br>tcsi:set_sys_resources | Get statistics on disk usage including state of mail system, short-term archive and recipient store |
| tcsiSetSystemResources | tcsi:set_sys_resources<br>tn:ok | Change disk configuration (sizes of MAIL, TECH and USER area) |
| tcsiGetTime | tcsi:set_get_time<br>tcsi:cl_time | Get current TCOSS time |
| tcsiSetTime | tcsi:set_time<br>tn:ok | Set TCOSS time |
| tcsiGetProgramVersion | tcsi:set_get_program_version<br>tcsi:cl_textstring | Get current TCOSS version |
| tcsiGetTcMsgId | tcsi:set_get_tc_msg_id<br>tcsi:cl_textstring | Returns a new unique TC Message ID |
| tcsiGetNumberOfSessions | tcsi:set_get_n_of_user_sessions<br>tcsi:cl_integer | Get number of currently open client sessions |
| tcsiGetNumberSeries | tcsi:set_get_number_series<br>tcsi:l_number_series | Get details of all TCOSS number series |
| tcsiSetNumberSerie | tcsi:set_number_serie<br>tn:ok | Change settings of a specific TCOSS number series |

| Function | Input/Output | Description |
|---|---|---|
| tcsiGetChannelStatus | tcsi:set_get_channel_status<br>tcsi:l_channel_status | Get a list of all TCOSS channels |
| tcsiSetChannelStatus | tcsi:set_channel_status<br>tn:ok | Stop or continue sending or reception on a TCOSS channel or trigger configuration reload by a channel |
| tcsiGetNodes | tcsi:set_get_nodes<br>tcsi:set_node | Get a list of all TCOSS nodes |
| tcsiNodeRestart | tcsi:set_node<br>tn:ok | Force to restart a TCOSS node |
| tcsiClearDisk | tcsi:set_disk_status<br>tn:ok | Clear a Disk in a Tandem server that is running in a desynchronized state |
| tcsiGetMessageFolder | tcsi:set_folder_ms<br>tcsi:set_folder_display | Get a (partial) list of documents in a message folder, the list may be filtered |
| tcsiGetUserFolder | tcsi:set_folder_us<br>tcsi:set_folder_display | Get a (partial) list of TCOSS users, the list may be filtered |
| tcsiGetRecipientFolder | tcsi:set_folder_rs<br>tcsi:set_folder_display | Get a (partial) list of recipients and distribution lists in the recipient store, the list may be filtered |
| tcsiGetServiceFolder | tcsi:set_folder_ss<br>tcsi:set_folder_display | Get a (partial) list of services, the list may be filtered |
| tcsiGetRegistrationFolder | tcsi:set_folder_regs<br>tcsi:set_folder_display | Get a (partial) list of registrations, the list may be filtered |
| tcsiGetLicenseFolder | tcsi:set_folder_license<br>tcsi:set_folder_display | Get a (partial) list of licenses, the list may be filtered |
| tcsiGetTimeZoneFolder | tcsi:set_folder_time_zone<br>tcsi:set_folder_display | Get a (partial) list of time zones, the list may be filtered |
| tcsiGetMailFolder | tcsi:set_folder_mail_sys<br>tcsi:set_folder_display | Get a (partial) inbox or outbox view of open send orders, the view may be filtered |
| tcsiGetMailArchiveFolder | tcsi:set_folder_mail_arc<br>tcsi:set_folder_display | Get a (partial) list of processed send orders from the short term archive, the list may be filtered |
| tcsiOpenMessage | tcsi:set_open_read<br>tcsi:set_entry_ms | Open a document from the message store |
| tcsiOpenUser | tcsi:set_open_read<br>tcsi:set_entry_us | Open a user profile |
| tcsiOpenRecipient | tcsi:set_open_read<br>tcsi:set_entry_rs | Open a recipient from the recipient store |
| tcsiOpenDistributionList | tcsi:set_open_read<br>tcsi:set_dl | Open a distribution list, returns all members of the list but does not resolve members of nested distribution lists |
| tcsiOpenLicense | tcsi:set_open_read<br>tcsi:set_entry_license | Open a TCOSS license |
| tcsiOpenRegistration | tcsi:set_open_read<br>tcsi:set_entry_regs | Open an existing registration or create a new registration with license check |

| Function | Input/Output | Description |
|---|---|---|
| tcsiOpenMail | tcsi:set_open_read<br>tcsi:set_entry_ms_mail | Open a message from the mail system |
| tcsiOpenMailArchive | tcsi:set_open_read<br>tcsi:set_entry_ms_mail_arc | Open a message from the short-term archive |
| tcsiOpenForTransfer | tcsi:set_open_read<br>tcsi:set_transfer_env | Get a message for outbound transfer from a queue |
| tcsiWriteObject | tcsi:set_write_obj<br>tn:ok | Stores a new version of a previously opened message, user profile, recipient, distribution list, service, license or transfer envelope, depending on the object type. In case of a message from the mail system only the mail entry is updated. In case of a transfer envelope the list of mail entries is updated. See also the tcsiSaveMessage, tcsiSaveUser, etc. group of functions which have a similar functionality but also return the updated entry. |
| tcsiWriteObjectAt | tcsi:set_write_at<br>tn:ok | Stores a new or existing object under a specified name. This function may be used to store documents in the message store, user profiles, recipients, distribution lists, services and licenses, depending on the object type. In the mail system it has the special function of posting a message. See also the tcsiSaveMessageAt, tcsiSaveUserAt, etc. group of functions which have a similar functionality but also return the new entry. |
| tcsiSaveMessage | tcsi:set_save_obj<br>tcsi:set_entry_ms | Stores a previously opened document from the message store and returns the updated entry |
| tcsiSaveUser | tcsi:set_save_obj<br>tcsi:set_entry_us | Stores a previously opened user profile and returns the updated entry |
| tcsiSaveRecipient | tcsi:set_save_obj<br>tcsi:set_entry_rs | Stores a previously opened recipient and returns the updated version |
| tcsiSaveService | tcsi:set_save_obj<br>tcsi:set_entry_ss | Stores a previously opened service and returns the updated version |
| tcsiSaveDistributionList | tcsi:set_save_obj<br>tcsi:set_dl | Stores a previously opened distribution list and returns the updated entry without the member list |
| tcsiSaveLicense | tcsi:set_save_obj<br>tcsi:set_entry_license | Stores a previously opened license and returns the updated version |
| tcsiSaveMail | tcsi:set_save_obj<br>tcsi:set_entry_ms_mail | Updates the mail entry of a previously opened message from the mail system and returns the updated entry |
| tcsiSaveMessageAt | tcsi:set_save_at<br>tcsi:set_entry_ms | Stores a document under the specified name in the message store and returns the entry |
| tcsiSaveUserAt | tcsi:set_save_at<br>tcsi:set_entry_us | Stores a user profile under the specified user ID and returns the entry |
| tcsiSaveRecipientAt | tcsi:set_save_at<br>tcsi:set_entry_rs | Stores a recipient under the specified recipient ID and returns the entry |

| Function | Input/Output | Description |
|---|---|---|
| tcsiSaveServiceAt | tcsi:set_save_at<br>tcsi:set_entry_ss | Stores a service under the specified name in the service store and returns the entry |
| tcsiSaveDistributionListAt | tcsi:set_save_at<br>tcsi:set_dl | Stores a distribution list under the specified list ID returns the updated entry without the member list |
| tcsiSaveLicenseAt | tcsi:set_save_at<br>tcsi:set_entry_license | Stores a new TCOSS license |
| tcsiSaveMailLog | tcsi:set_save_at<br>tcsi:set_entry_ms_mail_arc | Writes a user-defined log entry to the short-term archive and returns the entry |
| tcsiSaveMailTransfer | tcsi:set_save_at<br>tcsi:set_transfer_env | Save a transfer envelope. Depending on the type of message in the envelope this function can be used to route messages, route notifications or evaluate notifications |
| tcsiDeleteObject | tcsi:set_delete_obj<br>tn:ok | Delete an object from a permanent store. This function can be used to delete documents in the message store, user profiles, recipients, distribution lists, licenses, registrations and services, depending on the object type. Note that messages in the mail system can't be deleted with this function. They can be cancelled by opening the message, setting the state to “cancelled” and updating the entry with tcsiWriteObject or tcsiSaveMail. |

List of used namespaces

| Prefix | URI | Schema |
|---|---|---|
| tn | http://www.topcall.com/XMLSchema/2004/tn | result.xsd |
| tcsi | http://www.topcall.com/XMLSchema/2004/tcsi | tcsi.xsd |

## Known Issue with Visual Studio

When using tcsi functions from Visual Studio (observed with Visual Studio 2008), TWS returns the response with root element in lower case (e.g. “<set_entry_rs>”). However, Visual Studio expects upper case (e.g., “<SET_ENTRY_RS>”) instead and reports an invalid XML exception (“cannot deserialize the response”).

This is an issue with Visual Studio. As a workaround, add the XML attribute declaration in lower case before the class declaration generated by Visual Studio, e.g.:

```
[System.Xml.Serialization.XmlRoot(“set_entry_rs”)]
public partial class SET_ENTRY_RS {“
```

**Note** If you update the web references to TWS, your changes will be lost as Visual Studio overwrites them.

# TC/Archive Functions

These functions are defined in file tcarchive.wsdl. The table below lists all available functions with a short description. More details about these functions can be found in the TCSI manual.

| Function | Input/Output | Description |
|---|---|---|
| tcsiGetArchiveState | tcsi:set_get_state_archive<br>tcsi:set_state_archive | Read the archive state (active / stopped) plus some statistic values |
| tcsiSetArchiveState | tcsi:set_state_archive<br>tn:ok | Stop / restart archiving and archive search |
| tcsiSetVolume | tcsi:set_volume<br>tn:ok | Set offline volume path and actions on volume |
| tcsiGetVolumeFolder | tcsi:set_folder_arcvol<br>tcsi:set_folder_display | Read the archive volume overview |
| tcsiGetArchiveFolder | tcsi:set_folder_archive<br>tcsi:set_folder_display | Search messages in TC/Archive |
| tcsiOpenArchiveMessage | tcsi:set_open_read<br>tcsi:set_entry_archive | Open a message from TC/Archive |

List of used namespaces

| Prefix | URI | Schema |
|---|---|---|
| tn | http://www.topcall.com/XMLSchema/2004/tn | result.xsd |
| Tcsi | http://www.topcall.com/XMLSchema/2004/tcsi | tcsi.xsd |

# Maintenance Functions

These functions are defined in file master.wsdl. The table below lists all available functions with a short description.

| Function | Input/Output | Description |
|---|---|---|
| Ping | m:Ping<br>tn:ok | This function always returns ok. It can be used as simple test if the connection to TWS is available |
| GetState | m:GetState<br>m:CellState | Returns a state overview about all components |
| GetConfig | m:GetConfig<br>m:ComponentConfig | return the internal configuration of a component |

List of used namespaces

| Prefix | URI | Schema |
|---|---|---|
| tn | http://www.topcall.com/XMLSchema/2004/tn | result.xsd |
| m | http://www.topcall.com/XMLSchema/2005/master | master.xsd |

Chapter 8

# Customization

This section describes the customization of TWS.

## Exit Points

In addition to the configuration of TWS, the behavior of some web-service functions may be adapted to special customer needs via work-flow exit points. These exit points are located at well-defined positions during processing of the web-service calls. Depending on TWS configuration, the XML data at this point may be converted with customer-specific (XSLT or TSL) transformations. The actual transformation is stored in file Custom.xslt (typically located in directory c:\topcall\tws\00\Config). If this file does not exist, it will be created whenever configure.bat is called. If it already exists, it will not be changed during configuration and update of TWS.

**Remark**: Use of exit points requires knowledge of XML and XSLT. It should be used by experienced engineers only for problems that cannot be solved without customization.

### Location of Exit Points

The picture below shows some web-service calls (which may be customized by exits) and the KCS message server calls that they use. It provides an overview of functions that may be affected by exit points.


The web-service exit points on the left side (for functions: SendSimpleMessage, SendMessage, ReceiveMessage, ViewMessage and ConfirmMessage) may be used to modify the web service request/response from the client application. The corresponding input/output parameters are described in chapter *Micro Workflow Functions*.

The exit points on the right side (for functions: TcossPostMsg, TcossReceiveMsg, TcossGetContent and TcossGetMsgHeader) may be used to modify the request/response to/from the KCS message server (TCOSS). A short description of these functions can be found in chapter *Reference of TCOSS Exit Points*.

## How to Use Exit Points

The primary purpose of exit points is to modify the used XML data. In order to support development and troubleshooting, they can also be used to write the XML data at each point (before and after conversion) into a dedicated trace file. This debugging feature can be used with or without the conversion feature of exit points.

Exit points can be configured in the section “Exit Points for Customization” to one of the options shown below:

| Exit point configuration options | Description |
|---|---|
| <no exit> | Exit is inactive. The XML data is neither traced nor converted. |
| file-trace only | XML data at the exit will be written into a trace file. The data itself will not be modified. |
| Kofax-TSL | XML data will be converted with Kofax XSL transformation. |
| Kofax-TSL with file-trace | Like Kofax-TSL, but in addition, the XML data is written into trace files before and after conversion. |

| Exit point configuration options | Description |
|---|---|
| MS XSLT | XML data will be converted with Microsoft XSLT transformation. |
| MS XSLT with file trace | Like MS XSLT, but in addition, the XML data is written into trace files before and after conversion. |

The exits for `AnyFunction` (to be replaced by its real name, e.g. `SendMessage`) use the following names:

- `AnyFunction-Requ` defines the Exit-point for the request (input) data of function `AnyFunction`.
- `AnyFunction-Resp` defines the Exit-point for the response (output) data of function `AnyFunction`.

## Exit Point File Trace

The file-trace function writes the XML data to files in the TWS trace directory. The file name is build from the following two parts:

- Name of exit point (e.g. `SendMessage-Requ`)
- `1.xml` before conversion by exit or 2.xml after conversion by exit.

Examples:

- `C:\topcall\tws\00\trace\SendMessage-Requ1.xml` contains the input parameter for function SendMessage before it has been converted by the exit (original request from the client).
- `C:\topcall\tws\00\trace\SendMessage-Requ2.xml` contains the input parameter for function SendMessage after it has been converted by the exit (this is the request handled by TWS)

Additional remarks:

- If the file already exists, it will be overwritten. Thus you typically see the XML data from the last recent call.
- If the write operation fails for any reason, no data will be written and no error will be returned to the calling user (but the error can be found in the TWS message trace).
- If no conversion is configured, only the file-trace before conversion (e.g. SendMessage-Requ1.xml) will be created.
- If the used function supports an exit point for the request only and a file-trace is configured for the request, a file-trace for the response will be added as well.
- This trace should be used during development or troubleshooting only. It should not be used in a productive environment.

## Conversion with Kofax TSL (or MS XSLT) Engine

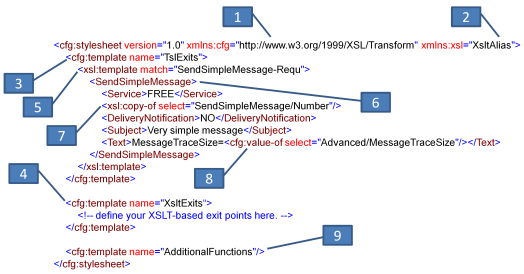
In order to activate the conversion via Kofax TSL (MS XSLT) engine you have to:

- Add a style sheet which matches the name of exit point to the template with name TslExits (or XsltExits) in the file config\Custom.xslt. See example below.
- Activate Kofax TSL (of MS XSLT) in the exit point configuration value.
- Restart TWS in order to use the new configuration.

**Important** The exit point transformation from Custom.xslt will be merged into the internal configuration used by TWS. Therefore, each modification in Custom.xslt requires a configuration refresh (start configuration tool and then restart TWS). Otherwise the modifications in Custom.xslt are not used!

## Content of Custom.xslt

Custom.xslt must be a valid XSLT style sheet. If you do not use any exits, the style sheet does not need to have any templates. A simple ExitPoint.xslt which converts the input to SendSimpleMessage via TSL workflow is shown below:




[1] Custom.xslt must be a valid XSLT style sheet which requires the name space URI http://www.w3.org/1999/XSL/Transform. It is recommended to use the name space prefix `cfg` to indicate that this style sheet will be applied during generation of the internal configuration for TWS.

[2] All XSLT style sheet elements which should be part of the exit point transformation at runtime must be declared in the name space `XsltAlias`. It will be transformed into the standard XSLT namespace http://www.w3.org/1999/XSL/Transform when the internal configuration will be written. It is recommended to use the prefix xsl.

[3] `cfg:template` with name `TslExits` contains all transformations using the Kofax TSL engines.

[4] `cfg:template` with the name `XsltExits` must be used for all transformations with Microsoft XSLT.

[5] Each configured exit requires an `xsl:template` which matches the name of the exit. If an exit is activated and no match is found, a default template which returns fault will be applied.

[6] The template gets the current request / response as a single input node with the name of the request / response. The template body must return the modified data as an output node (SendSimpleMessage in the example above). If you do not want to convert anything, you can use `<xsl:copy-of select=”.” />` to return the input without any modification.

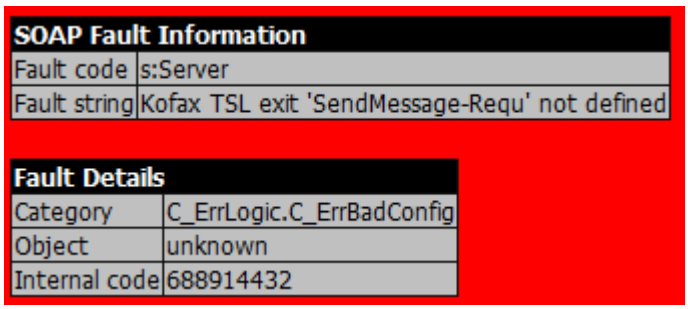
[7] XSLT elements within the name-space prefix `xsl` are used to generate output data based on the current input (web service request to `SendSimpleMessage` in the example above). You can use all transformations supported by the selected engine (either MS XSLT or Kofax TSL).

[8] XSLT element with the name-space prefix `cfg` is used to generate the output data based on the current configuration. The example shows how to merge the configured message trace size into the output. This transformation will be executed by AltovaXml which supports the XSLT 1.0 standard.

[9] The template `AdditionalFunctions` is used for custom web service functions as described in chapter Custom Web Services.

**Remarks**:

- If `<cfg:template>` with the correct name is not found, a default `<cfg:template>` from file `xcd/CustomDefault.xslt` (without exit point transformations) is used instead.
- If an exit point conversion is configured but no matching `<tsl:template>` is found, a default conversion into a fault as shown in the screen shot below is used.



| SOAP Fault Information | |
|---|---|
| Fault code | s:Server |
| Fault string | Kofax TSL exit 'SendMessage-Requ' not defined |

| Fault Details | |
|---|---|
| Category | C_ErrLogic.C_ErrBadConfig |
| Object | unknown |
| Internal code | 688914432 |

- It is recommended to call “configure.bat” from a command prompt in order to update the internal configuration after changes in `Custom.xslt`. The output should be checked for errors.

## Conversion Engines

This section describes the conversion engines.

### Kofax TSL

The Kofax TSL engine supports a small subset of XSLT 1.0 and some Kofax proprietary extension. Please contact Kofax for more information.

### Microsoft XSLT

The engine is based on the “XslCompiledTransform” class of the Microsoft .NET Framework 2.0. It is an XSLT processor that supports the XSLT 1.0 syntax. XSLT scripting and the XSLT document() function are enabled.

For more details see http://msdn.microsoft.com/en-us/library/system.xml.xsl.xslcompiledtransform.aspx.

With the command line tool nxslt2.exe it is possible to test the used style sheet. However, the ksa:invoke and ksa:login extension functions can’t be tested in the command line environment. The nxslt tool version 2.3 can be downloaded from: http://www.xmllab.net/downloads/nxslt/.

### XSLT Scripting

The <msxsl:script> element can be used to include scripts in the style sheet.

See http://msdn.microsoft.com/en-us/library/ms256042.aspx for a detailed description.

## KSA Extension functions

The Kofax Server Architecture extension functions belong to the namespace http://www.kofax.com/2010/ksa.

The “ksa:login” function does a login to a TCOSS server, the “ksa:invoke” function allows to call any function provided by any of the installed components. Both functions transparently use the credentials (user ID, password) that were provided with the original web service call into TWS.

“**ksa:login” function**

The “ksa:login” function is used to test the connection to a TCOSS server verifying path, user ID, password and TCOSS license. It invokes the “LoginOnly” function of the TCSI connector and returns a “t:ok” element on success or a “Fault” on failure.

Syntax: ksa:login( *target*) or ksa:login*( target*, *lictype*)

Parameters:

- target… string, target component name
- lictype… integer, license type

Returned xml if ok:

```
<t:ok xmlns:t="http://www.topcall.com/XMLSchema/2004/tn"></t:ok>
```

Returned xml on failure (example):

```
<s:Fault xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <faultcode>s:Server</faultcode>
  <faultstring>612 server temporarily busy</faultstring>
  <detail>
    <t:result xmlns:t="http://www.topcall.com/XMLSchema/2004/tn">
      <t:code>558891526</t:code>
      <t:info>612 server temporarily busy</t:info>
    </t:result>
  </detail>
</s:Fault>
```

**Example**:

```
<cfg:template name="XsltExits">
 <xsl:template match="SendSimpleMessage-Requ" xmlns:ksa="http://www.kofax.com/2010/
ksa">
  <xsl:variable name="login_resp" select="ksa:login('tcoss')"/>
  <xsl:choose>
   <xsl:when test="string($login_resp)=''">
    <!-- proceed with normal flow -->
    <xsl:copy-of select="*"/>
   </xsl:when>
   <xsl:otherwise>
    <!-- handle Fault -->
    <xsl:copy-of select="$login_resp"/>
   </xsl:otherwise>
  </xsl:choose>
 </xsl:template>
</cfg:template>
```

The test on the ksa:login response above builds on the fact that the <t:ok /> response element is empty, so its string value is the empty string, while any Fault has character content.

Alternatively one could use a test like

```
 <xsl:when test="local-name($login_resp/*)='ok'">
```

**“ksa:invoke” Function**

The “ksa:invoke” function is used to call a function in a component, for example to access TCOSS or to incorporate one of the standard work flows. It returns the response of the function or a Fault if the invoke fails.

Syntax: ksa:invoke( *target, request*) or ksa:invoke*( target*, *request*, *timeout*)

Parameters:

- target –string, target component name
- request – string or xml, simple or structured request
- timeout – integer, optional invoke timeout in seconds, default is 120

In case of a simple request without parameters (like the “set_get_time” or “set_get_tc_msg_id” functions of TCOSS) the local name of the request start element can be specified as string. In the general case the request is specified as xml structure referenced by an XPATH.

**Example of ksa:invoke with simple request**:

```
<cfg:template name="XsltExits">
  <xsl:template match="SendSimpleMessage-Requ" xmlns:ksa="http://www.kofax.com/2010/
ksa">
    <SendSimpleMessage>
      <xsl:copy-of select="SendSimpleMessage/Service"/>
      <xsl:copy-of select="SendSimpleMessage/Number"/>
      <Subject>ID: <xsl:value-of select="ksa:invoke('tcoss','set_get_tc_msg_id')"/></
Subject>
      <xsl:copy-of select="SendSimpleMessage/Text"/>
    </SendSimpleMessage>
  </xsl:template>
</cfg:template>
```

The above example inserts a TCOSS message ID into the subject field. Note that there is no error handling if the ksa:invoke fails (assuming that posting of the message will most likely also fail afterwards).

**Example of ksa:invoke with structured request**:

```
<cfg:template name="XsltExits">
  <xsl:template match="SendSimpleMessage-Requ" xmlns:tc="http://www.topcall.com/
XMLSchema/2004/tcsi" xmlns:ksa="http://www.kofax.com/2010/ksa">
    <xsl:variable name="openrequ">
      <tc:set_open_read>
        <tc:set_selector_entry.set_entry_ms>
          <tc:ts_tos_folder>+MAIL5V</tc:ts_tos_folder>
          <tc:ts_file_name>F<xsl:value-of select="SendSimpleMessage/Text"/></
tc:ts_file_name>
        </tc:set_selector_entry.set_entry_ms>
      </tc:set_open_read>
    </xsl:variable>
    <xsl:variable name="mess" select="ksa:invoke('tcoss',$openrequ,60)"/>
    <xsl:choose>
      <xsl:when test="$mess/tc:set_entry_ms">
        <SendSimpleMessage>
```

```
          <xsl:copy-of select="SendSimpleMessage/Service"/>
          <xsl:copy-of select="SendSimpleMessage/Number"/>
          <xsl:copy-of select="SendSimpleMessage/Subject"/>
          <Text>
            <xsl:value-of select="$mess/tc:set_entry_ms/tc:un_content.l_env_cont/
tc:set_blk_txt/tc:un_content.blk_txt"/>
          </Text>
        </SendSimpleMessage>
      </xsl:when>
      <xsl:otherwise><!-- return Fault -->
        <xsl:copy-of select="$mess"/>
      </xsl:otherwise>
    </xsl:choose>
```

The above example exit expects a file name in the “Text” field of the “SendSimpleMessage” request, opens the referenced file from the TCOSS FIS folder and inserts its text content into the message. Just to show how a timeout is specified a 60 seconds timeout is used for the TCOSS access (the default would be 120 seconds).

Here is another example of an exit which expects a Californian city name in the “Subject” field of the “SendSimpleMessage” request, calls a web service to determine the coordinates of the place, and returns the result of the lookup instead of posting a message to TCOSS:

```
<xsl:template match="SendSimpleMessage-Requ" xmlns:ksa="http://www.kofax.com/2010/ksa">
   <xsl:variable name="requ">
      <CallWebService>
         <Url>http://msrmaps.com/TerraService2.asmx</Url>
         <SoapAction>"http://msrmaps.com/ConvertPlaceToLonLatPt"</SoapAction>
            <Body>
               <ConvertPlaceToLonLatPt xmlns="http://msrmaps.com/">
                  <place>
                     <City><xsl:value-of select="SendSimpleMessage/Subject"/></City>
                     <State>California</State>
                     <Country>USA</Country>
                  </place>
               </ConvertPlaceToLonLatPt>
            </Body>
      </CallWebService>
   </xsl:variable>
   <xsl:copy-of select="ksa:invoke('http',$requ)"/>
</xsl:template>
```

## Reference of TCOSS Exit Points

This chapter provides a brief overview about some KCS server (TCOSS) functions that are mentioned in *Customization*. Since these functions may have different effects depending on the used parameters, a set of pseudo names is used instead of the real KCS message server calls (these are already described in *TCOSS Functions*).

### TcossGetDefaultOriginator

This function returns the address book entry of the current active user. Typical request structure:

```
<set_open_read >
  <set_entry.set_entry_ms>
    <set_selector_entry.set_entry_rs>
      <ts_section>+TECH</ts_section>
      <ts_recp_id>User</ts_recp_id> <!-- current user ID -->
    </set_selector_entry.set_entry_rs>
```

```
  </set_open_read>
</set_open_read>
```

## TcossPostMsg

Posts a new message to TCOSS. Typical request structure:

```
<set_write_at>
   <set_entry.set_entry_ms>
…
```

## 'TcossGetMsgId'

Returns a new unique message ID from the KCS message server. Request structure:

```
<set_get_tc_msg_id/>
```

## 'TcossReceiveMsg'

Gets messages from a Queue. Typical request structure:

```
<set_open_read>
  <set_selector_entry.set_selector_ms_mail>
    <int_max_entries>1</int_max_entries>
    <int_options>0</int_options>
    <ts_recipient>XX</ts_recipient>
    <int_aspects>0</int_aspects>
  </set_selector_entry.set_selector_ms_mail>
</set_open_read>
```

## 'TcossGetImageView'

Returns the image view of a message which is specified by CIF-Nr and CIF-Id. Typical request structure:

```
<set_open_read>
  <set_selector_entry.set_aspect_ms_mail>
    <int_cif_nr>12345"<int_cif_nr>
    <int_cif_id>33333"<int_cif_id>
    <int_aspects>15</int_aspects>  <!—9 means w/o cover -->
  </set_selector_entry.set_aspect_ms_mail>
</set_open_read>
```

## 'TcossGetContent'

Returns the content of a message which is specified by CIF-Nr and CIF-Id. Typical request structure:

```
<set_open_read>
  <set_selector_entry.set_aspect_ms_mail>
    <int_cif_nr>12345<int_cif_nr>
    <int_cif_id>33333<int_cif_id>
    <int_aspects>8</int_aspects>
  </set_selector_entry.set_aspect_ms_mail>
</set_open_read>
```

## 'TcossGetMsgHeader'

Returns the header information of an existing message. Typical request structure:

```
<set_open_read>
```

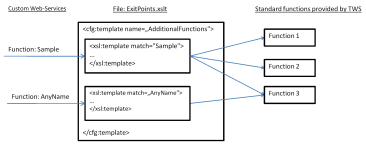
```
  <set_selector_entry.set_aspect_ms_mail>
    <int_cif_nr>12345"<int_cif_nr>
    <int_cif_id>33333"<int_cif_id>
    <int_aspects>3</int_aspects>
  </set_selector_entry.set_aspect_ms_mail>
</set_open_read>
```

### 'TcossConfirmMsg'

Updates a message which has been returned by web-service function TcossReceiveMsg.

# Custom Web Services

Custom workflows can be used to implement new web-service functions which are not part of standard TWS. Each web service call has to be implemented as a Kofax TSL template within Custom.xslt (is not modified by upgrades of TWS). The Kofax TSL template can include calls to all standard TWS web services. An overview is shown below:



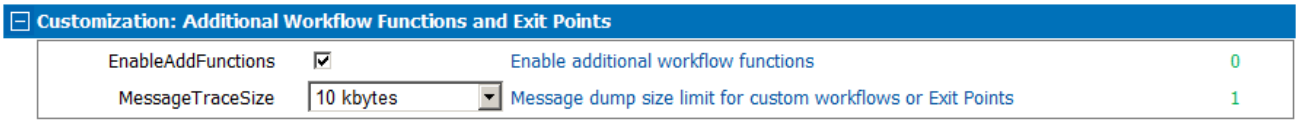

Custom web services may be used in the following cases:

- An existing web service with complex parameters (e.g. TCOSS functions described in 7.3) should be simplified because it is too complex for the calling application.
- Any activity which currently requires two or more web-service calls should be available with a single web service call.

TWS is shipped with a sample custom web service. It is recommended to modify this web service according to the functions needed by the customer.

## Activation of Custom Web Services

Custom web services have to be enabled in the configuration. In addition, it is possible to activate message trace for the custom web services (and exit points). This trace is recommended during development and troubleshooting. It traces both of the input/output from the calling application and the web services called by Kofax TSL template.




## Sample Web Service

TWS is installed with the following custom web service sample:

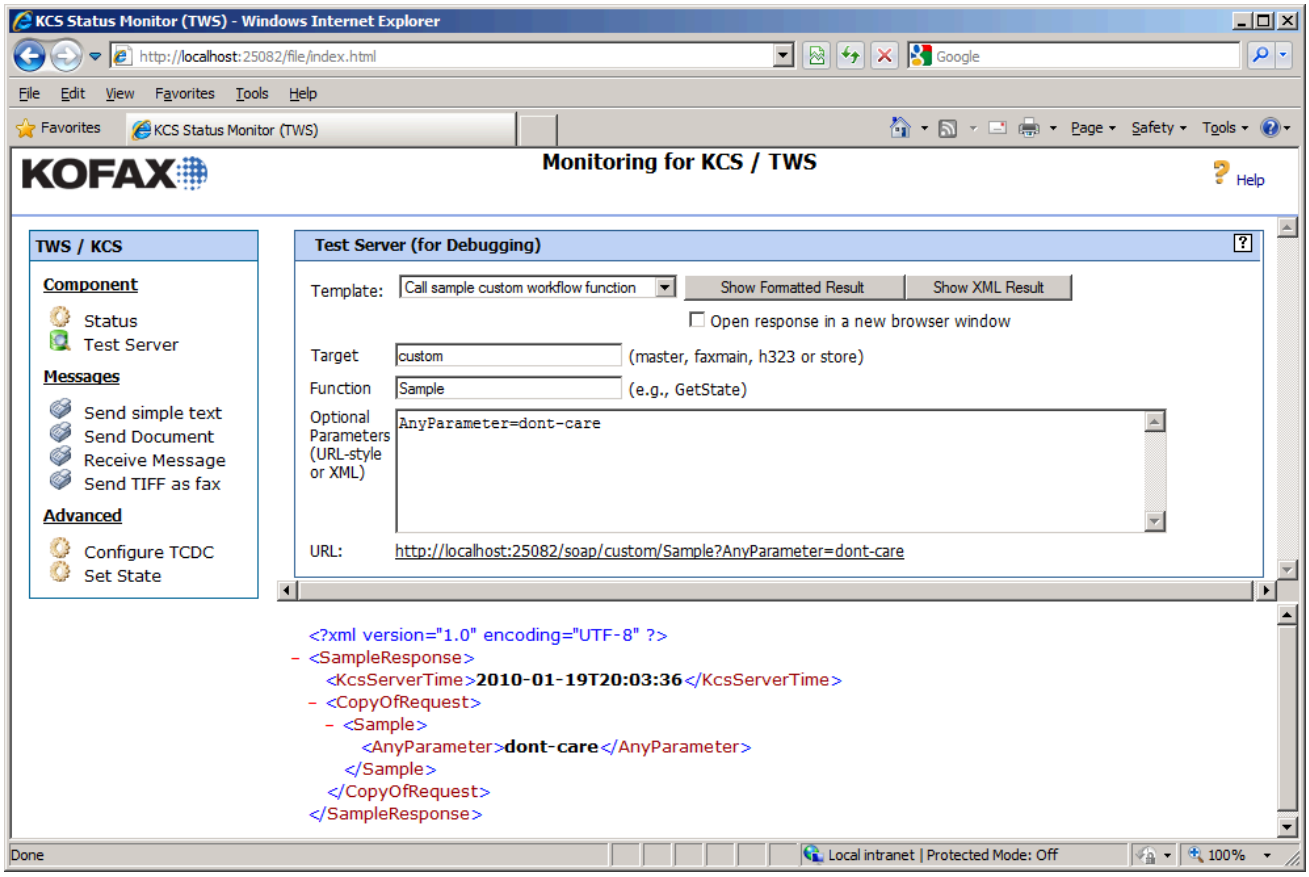
```
<cfg:stylesheet version="1.0" xmlns:cfg="http://www.w3.org/1999/XSL/Transform"
 xmlns:xsl="XsltAlias">
<cfg:template name="AdditionalFunctions" xmlns:tsl="http://www.topcall.com/2004/tsl">
    <xsl:template match="Sample">
      <!-- return the current TCOSS time. Save result in variable $time -->
      <xsl:variable name="time">
        <tsl:invoke tsl:target="tcoss">
          <tcsi:set_get_time tcsi:xmlns="http://www.topcall.com/XMLSchema/2004/tcsi"/>
        </tsl:invoke>
      </xsl:variable>
      <xsl:choose>
        <xsl:when test="$time/Fault">
          <xsl:copy-of select="$time"/> <!-- return the fault from TCOSS -->
        </xsl:when>
        <xsl:otherwise>
          <SampleResponse>
            <KcsServerTime><xsl:value-of select="$time"/></KcsServerTime>
            <CopyOfRequest>
              <xsl:copy-of select="."/> <!-- return the complete request -->
            </CopyOfRequest>
          </SampleResponse>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:template>
</cfg:template>
</cfg:stylesheet>
```

This web service tries to read to current time from the KCS server. If the call fails, the fault is returned without any modification. Otherwise, the server time and a copy of the request are returned.

Custom web services may be called either by

- HTTP get request with url=http[s]://{host}[:port]/soap/custom/{function-name}[?{parameters}].
- Web service call with url=http[s]://{host}[:port]/soap/custom; soap-action={function-name}.

The first method can be tested by using the test server function provided by the TWS web portal as shown in the screen shot below:




## Generation of WSDL Files

If the custom web service should be used by an application, most compiler based languages (e.g. C ++, C# or Java) require the WSDL (web service definition language) file for code generation during development. Since the templates that actually implement the custom web service function do not include a description of the input and output parameters, the following manual steps are required. If your application does not require a WSDL file, this chapter can be skipped.

To create a WSDL file for the sample described in the previous chapter:

- Copy the files `custom.xml` and `custom.xcd` from directory `api\template` to directory `api`.
- Run `configure.bat` in order to create/update the file `web\custom.wsdl`

To support your own custom functions:

- Add the required types for the input/output parameters to the schema file `api\custom.xsd`. It is recommended to use a schema editor (e.g. Altova XMLSpy).
- Add the new function, their input/output types, and an optional short description to `api\custom.xml`.
- Run configure.bat in order to create/update the file `web\custom.wsdl`.

You may optionally change the name-space uri (used in `custom.xsd` and `custom.xml`) which uniquely identifies the interface from http://www.kofax.com/2010/Custom to any other uri.

**Important**

- TWS does not check whether the created WSDL file correctly describes the custom workflows. Therefore, it is strongly recommended to test the web services with an application that imports the WSDL file.

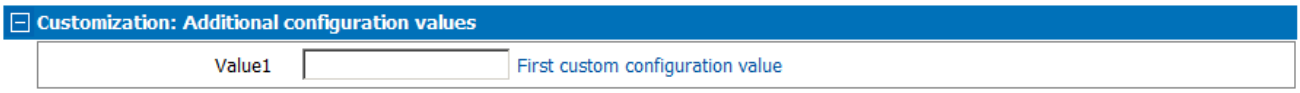
# Custom Configuration Values

Exit points and additional web service functions (as described above) can use all configuration values defined by TWS for SolutionConfig.xml. If an exit point of custom web service function requires any new configuration parameter they can be added to the global type `CustomConfigType` defined in the XML schema file `config\CustomConfig.xsd` (located in the folder `config`). Like `Custom.xslt` the file `CustomConfig.xsd` will be created from a template if it does not exist. An existing `CustomConfig.xsd` will never be modified.

The default CustomConfig.xsd is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
 attributeFormDefault="qualified">
  <xs:complexType name="CustomConfigType">
    <xs:annotation>
      <xs:documentation>Configuration of the KCS Connection</xs:documentation>
     </xs:annotation>
     <xs:sequence>
<!--
       <xs:element name="Value1" type="xs:normalizedString" minOccurs="0">
         <xs:annotation>
           <xs:documentation>First custom configuration value</xs:documentation>
         </xs:annotation>
       </xs:element>
-->
    </xs:sequence>
 </xs:complexType>
</xs:schema>
```

The template contains a comment with one configuration value sample. If you remove this comment you will get a new configuration section as shown in the screen shot below:




It is recommended to edit this file with an XML schema editor. You can use one of the following types:

| Type | Description |
|---|---|
| xs:Boolean | Will be displayed as a check box |
| xs:normalizedString | Standard input field. It may optionally be used with enumeration list. |

The following additional features are supported for string values:

- If a string contains an enumeration facet it will be displayed as list box in the configuration GUI

- The default values can be defined in the schema. They are used if the corresponding value does not exist in SolutionConfig.xml. If no default is specified in the schema, an empty string is used as default.

You can merge custom configuration value into your TSL or XSLT transformation via XPATH starting with Custom. An example exit point for SendSimpleMessage which uses Custom configuration value Value1 as service is shown in the example below:

```
<xsl:template match="SendSimpleMessage-Requ">
 <SendSimpleMessage>
    <Service><cfg:value-of select="Custom/Value1"/></Service>
    <xsl:copy-of select="SendSimpleMessage/Number"/>
    <xsl:copy-of select="SendSimpleMessage/Subject"/>
    <xsl:copy-of select="SendSimpleMessage/Text"/>
  </SendSimpleMessage>
</xsl:template>
```

# Examples

This section describes various examples.

## TSL Exits

SendSimpleMessage: Copy via template match and change the service element

```
<!-- Copy via template match and change the service element -->
<xsl:template match="Service" mode="SendSimpleMessage-Requ">
  <Service>TOPCALL</Service>
</xsl:template>
<xsl:template match="SendSimpleMessage-Requ">
  <xsl:apply-templates select="*" mode="SendSimpleMessage-Requ"/>
</xsl:template>
<xsl:template match="*" mode="SendSimpleMessage-Requ">
  <xsl:choose>
    <xsl:when test="*">
      <xsl:copy>
        <xsl:apply-templates select="*" mode="SendSimpleMessage-Requ"/>
      </xsl:copy>
    </xsl:when>
    <xsl:otherwise>
      <xsl:copy-of select="."/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

SendMessage: Copy via template match and change the cost center

```
<!-- Change the cost center -->
<xsl:template match="CostCenter" mode="SendMessage-Requ">
  <wf:CostCenter>SPEC-TSL</wf:CostCenter>
</xsl:template>
<!-- Copy the whole XML content recursivley -->
<xsl:template match="SendMessage-Requ">
  <xsl:apply-templates select="*" mode="SendMessage-Requ"/>
</xsl:template>
<xsl:template match="*" mode="SendMessage-Requ">
  <xsl:choose>
    <xsl:when test="*">
      <xsl:copy>
        <xsl:apply-templates select="*" mode="SendMessage-Requ"/>
```

```
      </xsl:copy>
    </xsl:when>
    <xsl:otherwise>
      <xsl:copy-of select="."/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

## XSLT Exits

SendMessage: Copy via template match and change the cost center, remove all department tags and add a company before the recipient name

```
<cfg:stylesheet version="1.0" xmlns:cfg="http://www.w3.org/1999/XSL/Transform"
xmlns:xsl="XsltAlias" xmlns:wf="http://www.topcall.com/2005/MicroWorkflows" >
<!-- Change the cost center of the header options -->
<xsl:template match="wf:SendMessage/wf:Header/wf:Options/wf:CostCenter"
              mode="SendMessage-Requ">
  <wf:CostCenter>SPEC-XSL</wf:CostCenter>
</xsl:template>
<!-- Remove all department tags -->
<xsl:template match="wf:Department" mode="SendMessage-Requ"/>
<!-- Add a company tag before recipient name -->
<xsl:template match="wf:SendMessage/wf:Header/wf:Recipient/wf:Name"
              mode="SendMessage-Requ">
  <wf:Company>A Company Name from XSLT exit</wf:Company>
  <xsl:copy>
    <xsl:apply-templates select="node()"/>
  </xsl:copy>
</xsl:template>
<!-- Copy the whole XML content recursivley -->
<xsl:template match="SendMessage-Requ">
  <xsl:apply-templates select="*" mode="SendMessage-Requ"/>
</xsl:template>
<xsl:template match="*" mode="SendMessage-Requ">
  <xsl:choose>
    <xsl:when test="*">
      <xsl:copy>
        <xsl:apply-templates select="*" mode="SendMessage-Requ"/>
      </xsl:copy>
    </xsl:when>
    <xsl:otherwise>
      <xsl:copy-of select="."/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

ReceiveMessage: Copy via template match and change the file name of the attachments, remove all company tags, add a new element before the originator title

```
<cfg:stylesheet version="1.0" xmlns:cfg="http://www.w3.org/1999/XSL/Transform"
xmlns:xsl="XsltAlias" xmlns:wf="http://www.topcall.com/2005/MicroWorkflows">
<!-- Change the file name of the attachments -->
<xsl:template match="wf:ReceiveMessageResponse/wf:SelectedContent/*/wf:LongFileName"
                  mode="ReceiveMessage-Resp">
  <wf:LongFileName>An XSL Exit-Generated File Name.docx</wf:LongFileName>
</xsl:template>
<!-- Remove all Company tags -->
<xsl:template match="wf:ReceiveMessageResponse/wf:Header/wf:MessageEntry/wf:Originator/
wf:Company"
                  mode="ReceiveMessage-Resp"/>
<!-- Add a new tag before Originator Title;
```

```
Note: This is not defined by the namespace and therefore might not work with all tools
-->
<xsl:template match="wf:ReceiveMessageResponse/wf:Header/wf:MessageEntry/wf:Originator/
wf:Title"
                  mode="ReceiveMessage-Resp">
  <MyOwnPrivateExitAttribute>Here is some information coming from the XSL exit</
MyOwnPrivateExitAttribute>
  <xsl:copy>
    <xsl:apply-templates select="node()"/>
  </xsl:copy>
</xsl:template>
<!-- Copy the whole XML content recursivley -->
<xsl:template match="ReceiveMessage-Resp">
  <xsl:apply-templates select="*" mode="ReceiveMessage-Resp"/>
</xsl:template>
<xsl:template match="*" mode="ReceiveMessage-Resp">
  <xsl:choose>
    <xsl:when test="*">
      <xsl:copy>
        <xsl:apply-templates select="*" mode="ReceiveMessage-Resp"/>
      </xsl:copy>
    </xsl:when>
    <xsl:otherwise>
      <xsl:copy-of select="."/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
...
```

TcossGetMsgHeader: Copy via template match and change ts_company

```
<cfg:stylesheet version="1.0" xmlns:cfg="http://www.w3.org/1999/XSL/Transform"
xmlns:xsl="XsltAlias" xmlns:c="http://www.topcall.com/XMLSchema/2004/tcsi">
...
<!-- Change the company -->
<xsl:template match="c:ts_company" mode="TcossGetMsgHeader-Resp">
  <c:ts_company>Company set by TcossGetMsgHeader XSL exit</c:ts_company>
</xsl:template>
<!-- Copy the whole XML content recursivley -->
<xsl:template match="TcossGetMsgHeader-Resp">
  <xsl:apply-templates select="*" mode="TcossGetMsgHeader-Resp"/>
</xsl:template>
<xsl:template match="*" mode="TcossGetMsgHeader-Resp">
  <xsl:choose>
    <xsl:when test="*">
      <xsl:copy>
        <xsl:apply-templates select="*" mode="TcossGetMsgHeader-Resp"/>
      </xsl:copy>
    </xsl:when>
    <xsl:otherwise>
      <xsl:copy-of select="."/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
...
```

## XSLT Exits with Invoke

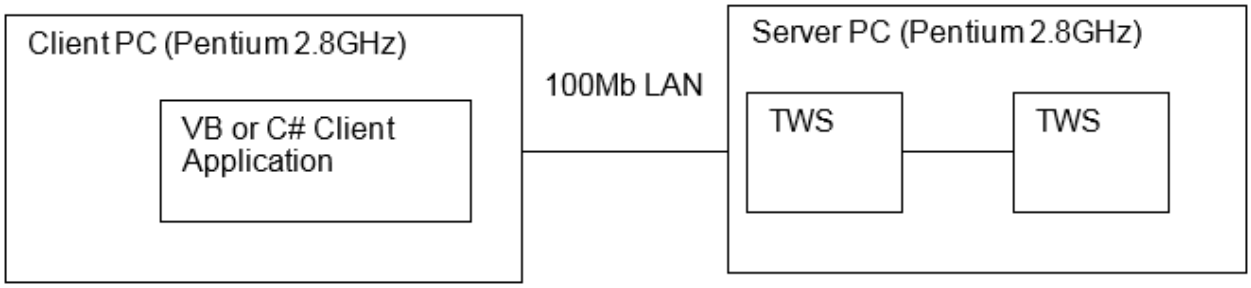
Example of the ksa:invoke function with SendMessage. The text of the message is interpreted as TCOSS FIS folder file name and set to the content of that file. The message is copied via template match, only the text body part is changed; in case of failure, the error text is written to the text body. Additionally the cost center is set to a fixed value.

```
<xsl:template match="wf:SendMessage/wf:Content/wf:Attachment/wf:TextContent"
                  mode="SendMessage-Requ"
xmlns:tc="http://www.topcall.com/XMLSchema/2004/tcsi"
xmlns:ksa="http://www.kofax.com/2010/ksa"
    xmlns:wf="http://www.topcall.com/2005/MicroWorkflows">
  <xsl:variable name="openrequ">
    <tc:set_open_read>
      <tc:set_selector_entry.set_entry_ms>
        <tc:ts_tos_folder>+MAIL5V</tc:ts_tos_folder>
        <tc:ts_file_name>
          F<xsl:value-of select="."/>
        </tc:ts_file_name>
      </tc:set_selector_entry.set_entry_ms>
    </tc:set_open_read>
  </xsl:variable>
  <xsl:variable name="mess" select="ksa:invoke('tcoss',$openrequ,60)"/>
  <xsl:choose>
    <xsl:when test="$mess/tc:set_entry_ms">
      <TextContent>
        <xsl:value-of select="$mess/tc:set_entry_ms/tc:un_content.l_env_cont/
tc:set_blk_txt/tc:un_content.blk_txt"/>
      </TextContent>
    </xsl:when>
    <xsl:otherwise>
      <!-- return Fault -->
      <TextContent>
        <xsl:copy-of select="$mess"/>
      </TextContent>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
<xsl:template match="wf:SendMessage/wf:Header/wf:Options/wf:CostCenter"
                  mode="SendMessage-Requ">
  <wf:CostCenter>SPEC-XSL</wf:CostCenter>
</xsl:template>
<xsl:template match="SendMessage-Requ">
  <xsl:apply-templates select="*" mode="SendMessage-Requ"/>
</xsl:template>
<xsl:template match="*" mode="SendMessage-Requ">
  <xsl:choose>
    <xsl:when test="*">
      <xsl:copy>
        <xsl:apply-templates select="*" mode="SendMessage-Requ"/>
      </xsl:copy>
    </xsl:when>
    <xsl:otherwise>
      <xsl:copy-of select="."/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Chapter 9

# Performance

The performance of TWS is similar to an application using TFC. TWS can handle up to 10 concurrent connections. Some very rough values have been measured using the following configuration:




## Test 1 – Send Simple Text Message

This test was made with a simple C# program (like the sample code in shown in chapter Send a Simple Text Message). The program uses TWS to send sort text messages. Here are the performance results:

| | |
|---|---|
| Messages per hour: | 37.000 |
| CPU load by client application: | 1% |
| CPU load by TWS on server: | 50% |
| CPU load by TCOSS on server: | 40% |
| Disk usage on TCOSS server: | 8% |

## Test 2 – Receive Message as PDF

This test was made with a Visual Basic program (like the sample code in shown in chapter Send a Simple Text Message). The program gets the next message from a queue and converts it into PDF. The PDF data will be stored in a file on the client and then the reception of the message will be confirmed. Here are the performance results:

| | | |
|---|---|---|
| Message: | Short Text message | 1 Page Image (125k TCI code) |
| Messages per hour: | 1565 | 4615 |
| CPU load by client application: | 50% | 44% |

| | | |
|---|---|---|
| CPU load by TWS on server: | 28% | 43% |
| CPU load by TCOSS on server: | 10% | 6% |
| Disk usage on TCOSS server: | 1.5% | 1.8% |

**Note** The test was made with short text messages (as sent in the test above) and a complex image page. The most CPU intensive task within TWS is the conversion of the message into PDF. It is currently implemented using TCSI and TCIMGIO libraries.

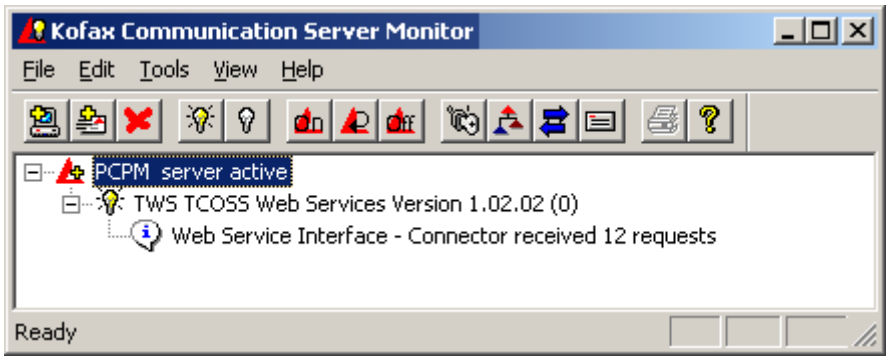
Chapter 10

# Troubleshooting

This section describes how to troubleshoot errors.

## KCS Monitor

The KCS Monitor may be used to verify that the web service processes are running:




## Activation of Traces

If you encounter any problems with TWS, please use the following trace options:

- Trace messages between components
- General trace for TWS
- TCSI and TCTI traces

In most cases, the problem can be located by tracing the messages between the components.

To change TWS trace options, launch the TWS configuration utility and update the settings in the Advanced tab. Refer to chapter *Installation Procedure* for more information.

**Note** Whenever you change the trace settings, TWS must be restarted.

### Trace Messages Between Components

For troubleshooting it is recommended to set the *Message Trace Size* to 10000. This means that the first 10000 bytes of each XML message being sent between TWS internal components will be traced.

For productive use set the *Message Trace Size* to its default value of 1 which means that only a single trace line will be written for each message.

The trace output will be written into the files C:\topcall\tws\00\trace\tws_xxx.trc where xxx is the actual trace file number.

## Example

A plain trace when sending a simple message is shown below. It starts with the message that has been received by the http connector (id=0). It will be sent to the micro workflow engine (id=3, name=tsl)

```
13/19:22:30.621 (0e04/0fa0) Dump-Req: Message (298 byte) from 0(no_name) ---> 3(tsl)
13/19:22:30.621 (0e04/0fa0) Dump-Req:  <s:Envelope xmlns:s="http://schemas.xmlsoap.org/
soap/
envelope/" xmlns:t="http://www.topcall.com/XMLSchema/2004/tn">
13/19:22:30.621 (0e04/0fa0) Dump-Req:    <s:Header>
13/19:22:30.621 (0e04/0fa0) Dump-Req:      <t:type>1</t:type>
13/19:22:30.621 (0e04/0fa0) Dump-Req:      <t:target>tsl</t:target>
13/19:22:30.621 (0e04/0fa0) Dump-Req:    </s:Header>
13/19:22:30.621 (0e04/0fa0) Dump-Req:    <s:Body>
13/19:22:30.621 (0e04/0fa0) Dump-Req:      <SendSimpleMessage>
13/19:22:30.621 (0e04/0fa0) Dump-Req:        <Service>FREE</Service>
13/19:22:30.621 (0e04/0fa0) Dump-Req:        <Number>GUEST:</Number>
13/19:22:30.621 (0e04/0fa0) Dump-Req:        <Subject>Test 4</Subject>
13/19:22:30.621 (0e04/0fa0) Dump-Req:        <Text>This is a simple message</Text>
13/19:22:30.621 (0e04/0fa0) Dump-Req:      </SendSimpleMessage>
13/19:22:30.621 (0e04/0fa0) Dump-Req:    </s:Body>
13/19:22:30.621 (0e04/0fa0) Dump-Req:  </s:Envelope>
```

In the next step, the Micro Workflow Engine (id=3, name=tsl) will send a command to the TCOSS component (id=1, name=tcoss) to get the name of the user that is currently logged in.

```
13/19:22:30.637 (0e04/0494) Dump-Req: Message (242 byte) from 3(tsl) ---> 1(tcoss)
13/19:22:30.637 (0e04/0494) Dump-Req:  <s:Envelope
xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:t="http://www.topcall.com/XMLSchema/2004/tn">
13/19:22:30.637 (0e04/0494) Dump-Req:    <s:Header>
13/19:22:30.637 (0e04/0494) Dump-Req:      <t:type>1</t:type>
13/19:22:30.637 (0e04/0494) Dump-Req:      <t:target>tcoss</t:target>
13/19:22:30.637 (0e04/0494) Dump-Req:    </s:Header>
13/19:22:30.637 (0e04/0494) Dump-Req:    <s:Body
xmlns="http://www.topcall.com/XMLSchema/2004/tcsi">
13/19:22:30.637 (0e04/0494) Dump-Req:      <GetUserId></GetUserId>
13/19:22:30.637 (0e04/0494) Dump-Req:    </s:Body>
13/19:22:30.637 (0e04/0494) Dump-Req:  </s:Envelope>
13/19:22:30.637 (0e04/0494) Dump-Rsp: Message (234 byte) from 1(tcoss) ---> 3(tsl)
13/19:22:30.637 (0e04/0494) Dump-Rsp:  <s:Envelope
xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:t="http://www.topcall.com/XMLSchema/2004/tn">
13/19:22:30.637 (0e04/0494) Dump-Rsp:    <s:Header>
13/19:22:30.637 (0e04/0494) Dump-Rsp:      <t:type>2</t:type>
13/19:22:30.637 (0e04/0494) Dump-Rsp:    </s:Header>
13/19:22:30.637 (0e04/0494) Dump-Rsp:    <s:Body
xmlns:c="http://www.topcall.com/XMLSchema/2004/tcsi">
13/19:22:30.637 (0e04/0494) Dump-Rsp:      <c:ts_user_id>guest</c:ts_user_id>
13/19:22:30.637 (0e04/0494) Dump-Rsp:    </s:Body>
13/19:22:30.637 (0e04/0494) Dump-Rsp:  </s:Envelope>
```

Next, the Micro Workflow Engine (id=3, name=tsl) will sent a command to the TCOSS component (id=1, name=tcoss) to open to address book entry of the logged-in user.

```
13/19:22:30.637 (0e04/0494) Dump-Req: Message (329 byte) from 3(tsl) ---> 1(tcoss)
```

```
13/19:22:30.637 (0e04/0494) Dump-Req:  <s:Envelope
xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:t="http://www.topcall.com/XMLSchema/2004/tn">
13/19:22:30.637 (0e04/0494) Dump-Req:    <s:Header>
13/19:22:30.637 (0e04/0494) Dump-Req:      <t:type>1</t:type>
13/19:22:30.637 (0e04/0494) Dump-Req:      <t:target>tcoss</t:target>
13/19:22:30.637 (0e04/0494) Dump-Req:    </s:Header>
13/19:22:30.637 (0e04/0494) Dump-Req:    <s:Body
xmlns="http://www.topcall.com/XMLSchema/2004/tcsi">
13/19:22:30.637 (0e04/0494) Dump-Req:      <set_open_read>
13/19:22:30.637 (0e04/0494) Dump-Req:        <set_selector_entry.set_entry_rs>
13/19:22:30.637 (0e04/0494) Dump-Req:          <ts_section>+TECH</ts_section>
13/19:22:30.637 (0e04/0494) Dump-Req:          <ts_recp_id>guest</ts_recp_id>
13/19:22:30.637 (0e04/0494) Dump-Req:        </set_selector_entry.set_entry_rs>
13/19:22:30.637 (0e04/0494) Dump-Req:      </set_open_read>
13/19:22:30.637 (0e04/0494) Dump-Req:    </s:Body>
13/19:22:30.637 (0e04/0494) Dump-Req:  </s:Envelope>
13/19:22:30.668 (0e04/0494) Dump-Rsp: Message (610 byte) from 1(tcoss) ---> 3(tsl)
13/19:22:30.668 (0e04/0494) Dump-Rsp:  <s:Envelope
xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:t="http://www.topcall.com/XMLSchema/2004/tn">
13/19:22:30.668 (0e04/0494) Dump-Rsp:    <s:Header>
13/19:22:30.668 (0e04/0494) Dump-Rsp:      <t:type>2</t:type>
13/19:22:30.668 (0e04/0494) Dump-Rsp:    </s:Header>
13/19:22:30.668 (0e04/0494) Dump-Rsp:    <s:Body
xmlns:c="http://www.topcall.com/XMLSchema/2004/tcsi">
13/19:22:30.668 (0e04/0494) Dump-Rsp:      <c:set_entry_rs>
13/19:22:30.668 (0e04/0494) Dump-Rsp:        <c:int_file_id>69529035</c:int_file_id>
13/19:22:30.668 (0e04/0494) Dump-Rsp:        <c:ts_section>+TECH</c:ts_section>
13/19:22:30.668 (0e04/0494) Dump-Rsp:        <c:ts_recp_id>GUEST</c:ts_recp_id>
13/19:22:30.668 (0e04/0494) Dump-Rsp:        <c:int_type>1</c:int_type>
13/19:22:30.668 (0e04/0494) Dump-Rsp:        <c:int_usage_count>0</c:int_usage_count>
13/19:22:30.668 (0e04/0494) Dump-Rsp:        <c:ts_fullname>Guest User</c:ts_fullname>
13/19:22:30.668 (0e04/0494) Dump-Rsp:        <c:int_dirsync_allowed>0</
c:int_dirsync_allowed>
13/19:22:30.668 (0e04/0494) Dump-Rsp:        <c:int_ownertype>0</c:int_ownertype>
13/19:22:30.668 (0e04/0494) Dump-Rsp:        <c:l_full_addr>
13/19:22:30.668 (0e04/0494) Dump-Rsp:          <c:set_full_address>
13/19:22:30.668 (0e04/0494) Dump-Rsp:            <c:ts_service>TOPCALL</c:ts_service>
13/19:22:30.668 (0e04/0494) Dump-Rsp:            <c:int_active>1</c:int_active>
13/19:22:30.668 (0e04/0494) Dump-Rsp:            <c:un_public_address.set_tc_address>
13/19:22:30.668 (0e04/0494) Dump-Rsp:              <c:ts_tc_userid>GUEST</
c:ts_tc_userid>
13/19:22:30.668 (0e04/0494) Dump-Rsp:            </c:un_public_address.set_tc_address>
13/19:22:30.668 (0e04/0494) Dump-Rsp:          </c:set_full_address>
13/19:22:30.668 (0e04/0494) Dump-Rsp:        </c:l_full_addr>
13/19:22:30.668 (0e04/0494) Dump-Rsp:      </c:set_entry_rs>
13/19:22:30.668 (0e04/0494) Dump-Rsp:    </s:Body>
13/19:22:30.668 (0e04/0494) Dump-Rsp:  </s:Envelope>
```

Then, the Micro Workflow Engine (id=3, name=tsl) will post the envelope to the TCOSS component (id=1, name=tcoss). The line “dump truncated (BufferSize=2000)“ indicates that the XML trace of the message exceeded that configured buffer size. It has been truncated after 2000 bytes. If you want to have a complete trace, the Message Trace Size has to be increased.

```
13/19:22:30.668 (0e04/0494) Dump-Req: Message (1100 byte) from 3(tsl) ---> 1(tcoss)
13/19:22:30.668 (0e04/0494) Dump-Req:  <s:Envelope
xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:t="http://www.topcall.com/XMLSchema/2004/tn">
13/19:22:30.668 (0e04/0494) Dump-Req:    <s:Header>
13/19:22:30.668 (0e04/0494) Dump-Req:      <t:type>1</t:type>
13/19:22:30.668 (0e04/0494) Dump-Req:      <t:target>tcoss</t:target>
13/19:22:30.668 (0e04/0494) Dump-Req:    </s:Header>
```

```
13/19:22:30.668 (0e04/0494) Dump-Req:    <s:Body
xmlns="http://www.topcall.com/XMLSchema/2004/tcsi">
13/19:22:30.668 (0e04/0494) Dump-Req:      <set_write_at>
13/19:22:30.668 (0e04/0494) Dump-Req:        <set_entry.set_entry_ms>
13/19:22:30.668 (0e04/0494) Dump-Req:          <un_content.l_env_cont>
13/19:22:30.668 (0e04/0494) Dump-Req:            <set_header>
13/19:22:30.668 (0e04/0494) Dump-Req:
<int_priority_cc>49</int_priority_cc>
13/19:22:30.668 (0e04/0494) Dump-Req:
<int_priority_to>50</int_priority_to>
13/19:22:30.668 (0e04/0494) Dump-Req:
<int_termination>81</int_termination>
13/19:22:30.668 (0e04/0494) Dump-Req:              <ts_ref>Test 4</ts_ref>
13/19:22:30.668 (0e04/0494) Dump-Req:
<set_entry_rs_originator.set_entry_rs>
13/19:22:30.668 (0e04/0494) Dump-Req:
<int_file_id>69529035</int_file_id>
13/19:22:30.668 (0e04/0494) Dump-Req:                <ts_section>+TECH</ts_section>
13/19:22:30.668 (0e04/0494) Dump-Req:                <ts_recp_id>GUEST</ts_recp_id>
13/19:22:30.668 (0e04/0494) Dump-Req:                <int_type>1</int_type>
13/19:22:30.668 (0e04/0494) Dump-Req:
<int_usage_count>0</int_usage_count>
13/19:22:30.668 (0e04/0494) Dump-Req:                <ts_fullname>Guest
User</ts_fullname>
13/19:22:30.668 (0e04/0494) Dump-Req:
<int_dirsync_allowed>0</int_dirsync_allowed>
13/19:22:30.668 (0e04/0494) Dump-Req:
<int_ownertype>0</int_ownertype>
13/19:22:30.668 (0e04/0494) Dump-Req:                <l_full_addr>
13/19:22:30.668 (0e04/0494) Dump-Req:                  <set_full_address>
13/19:22:30.668 (0e04/0494) Dump-Req:
<ts_service>TOPCALL</ts_service>
13/19:22:30.668 (0e04/0494) Dump-Req:                    <int_active>1</int_active>
13/19:22:30.668 (0e04/0494) Dump-Req:
<un_public_address.set_tc_address>
13/19:22:30.668 (0e04/0494) Dump-Req:
<ts_tc_userid>GUEST</ts_tc_userid> 13/19:22:30.668 (0e04/0494) Dump-Req:

</un_public_address.set_tc_address>
13/19:22:30.668 (0e04/0494) Dump-Req:                  </set_full_address>
13/19:22:30.668 (0e04/0494) Dump-Req:                </l_full_addr>
13/19:22:30.668 (0e04/0494) Dump-Req:
</set_entry_rs_originator.set_entry_rs>
13/19:22:30.668 (0e04/0494) Dump-Req:              <l_recipients>
13/19:22:30.668 (0e04/0494) Dump-Req:                <set_entry_rs>
13/19:22:30.668 (0e04/0494) Dump-Req:                  <int_active>1</int_active>
13/19:22:30.668 (0e04/0494) Dump-Req:
<int_del_type>1</int_del_type>
13/19:22:30.668 (0e04/0494) Dump-Req:                  <l_full_addr>
13/19:22:30.668 (0e04/0494) Dump-Req:                    <set_full_address>
13/19:22:30.668 (0e04/0494) Dump-Req:
<int_active>1</int_active>
13/19:22:30.668 (0e04/0494) Dump-Req:
<ts_service>FREE</ts_service> 13/19:22:30.668 (0e04/0494) Dump-Req:

<un_public_address.set_free_address>
13/19:22:30.668 (0e04/0494) Dump-Req:
<ts_free_addr>GUEST:</ts_free_addr> 13/19:22:30.668 (0e04/0494) Dump-Req:

</un_public_address.set_free_address>
13/19:22:30.668 (0e04/0494) Dump-Req:                    </set_full_address>
13/19:22:30.668 (0e04/0494) Dump-Req: dump truncated (BufferSize=2000)
13/19:22:30.731 (0e04/0494) Dump-Rsp: Message (170 byte) from 1(tcoss) ---> 3(tsl)
13/19:22:30.731 (0e04/0494) Dump-Rsp:  <s:Envelope
```

```
xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:t="http://www.topcall.com/XMLSchema/2004/tn">
13/19:22:30.731 (0e04/0494) Dump-Rsp:    <s:Header>
13/19:22:30.731 (0e04/0494) Dump-Rsp:      <t:type>2</t:type>
13/19:22:30.731 (0e04/0494) Dump-Rsp:    </s:Header>
13/19:22:30.731 (0e04/0494) Dump-Rsp:    <s:Body>
13/19:22:30.731 (0e04/0494) Dump-Rsp:      <t:ok></t:ok>
13/19:22:30.731 (0e04/0494) Dump-Rsp:    </s:Body>
13/19:22:30.731 (0e04/0494) Dump-Rsp:  </s:Envelope>
```

In the last step, the response from TCOSS will be sent to the http connection (id=0) which will return the response via http to the client.

```
13/19:22:30.731 (0e04/0fa0) Dump-Rsp: Message (170 byte) from 3(tsl) ---> 0(no_name)
13/19:22:30.731 (0e04/0fa0) Dump-Rsp:  <s:Envelope
xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:t="http://www.topcall.com/XMLSchema/2004/tn">
13/19:22:30.731 (0e04/0fa0) Dump-Rsp:    <s:Header>
13/19:22:30.731 (0e04/0fa0) Dump-Rsp:      <t:type>2</t:type>
13/19:22:30.731 (0e04/0fa0) Dump-Rsp:    </s:Header>
13/19:22:30.731 (0e04/0fa0) Dump-Rsp:    <s:Body>
13/19:22:30.731 (0e04/0fa0) Dump-Rsp:      <t:ok></t:ok>
13/19:22:30.731 (0e04/0fa0) Dump-Rsp:    </s:Body>
13/19:22:30.731 (0e04/0fa0) Dump-Rsp:  </s:Envelope>
```

## General Trace for TWS

This trace is controlled by the configuration value Trace Level. It can be set in a range between 0 and 255. Higher levels produce more trace output. This trace is intended for developers only. It is not recommended to be used for trouble shooting.

The trace output will be written into the files C:\topcall\tws\00\trace\tws_xxx.trc where xxx is the actual trace file number.

## TCSI and TCTI Traces

This trace will be controlled by the following registry values as described for existing KCS products.

| | |
|---|---|
| HKLM\Software\TOPCALL\TWS00\MaxTraceFiles | number of trace files |
| HKLM\Software\TOPCALL\TWS00\MaxTraceFileSize | max. Size of trace files |
| HKLM\Software\TOPCALL\TWS00\AppendTrace | enable append to trace files |
| HKLM\Software\TOPCALL\TWS00\TCSI\DebugLevel | TCSI Trace level |
| HKLM\Software\TOPCALL\TWS00\TCTI\TraceLevel | TCTI Trace level |

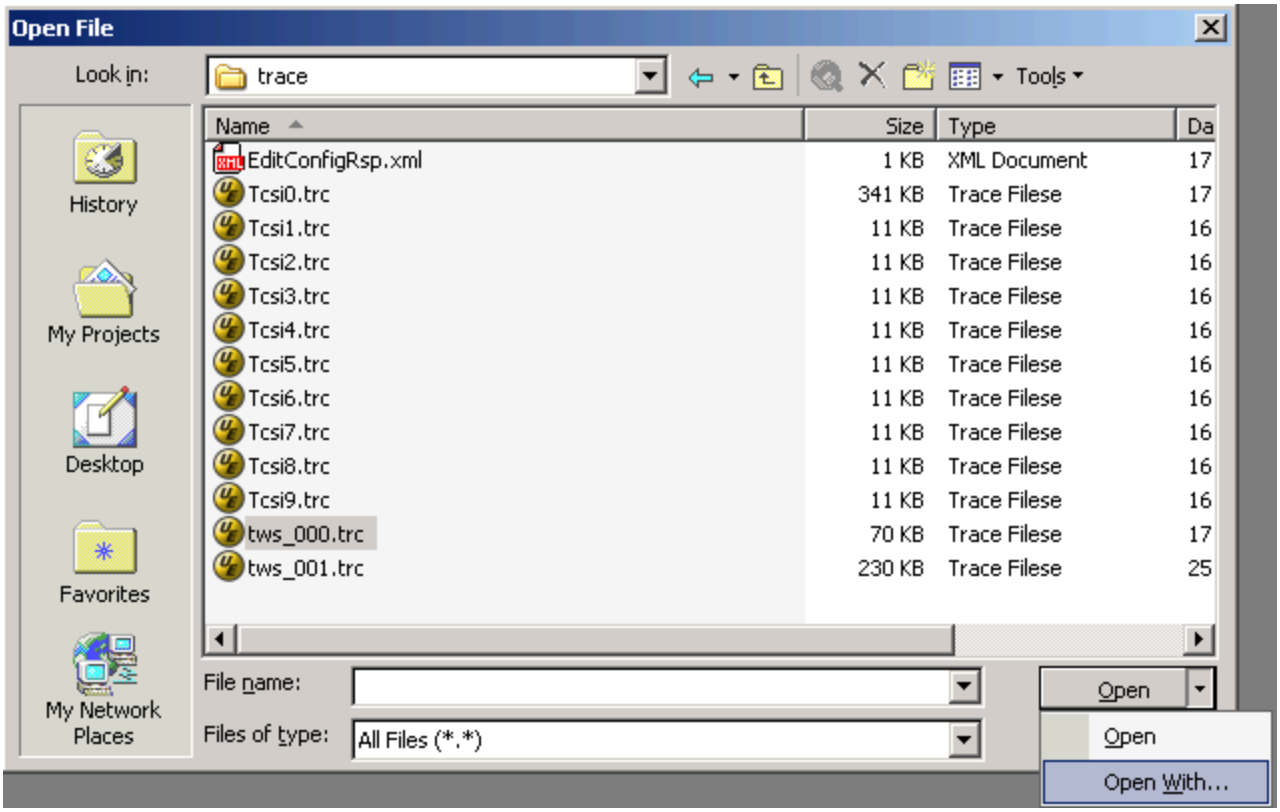
The trace output will be written into files C:\topcall\tws\00\trace\tcsi*.trc.

# View UTF-8 Traces

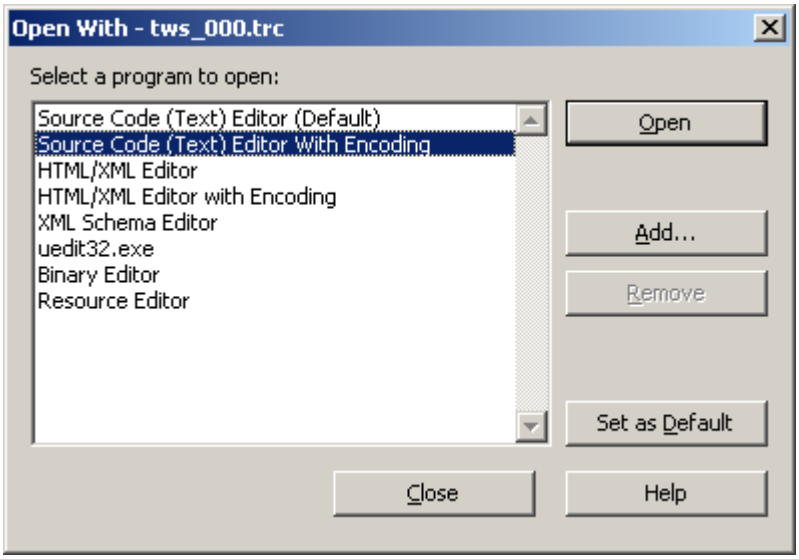
All traces generated by TWS (files tws*.trc) contain UTF-8 encoded text. If any standard text editor is used, the text files will be interpreted using the current Windows code page instead of UTF-8. If special

characters or vowel mutations are used, the traces files should be viewed with any UTF-8 editor. This chapter contains a description how to open the traces with Microsoft Visual Studio .net.

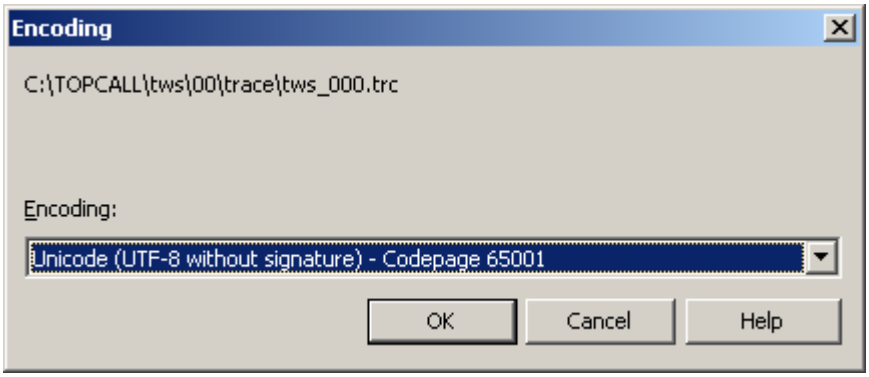
1. Open the Open File window.
2. Select the file you want to open and click **Open With…**




3. Select Source Code (Text) With Encoding and click **Open**.



4. As your encoding, select “Unicode (UTF-8 without signature) – Codepage 65001” and click OK.




# Frequent Errors

This chapter describes some errors that are likely to happen. If you have any problems it is recommended to make all checks listed below from top to down until you find the problem. In the right side some corrective actions for the most likely error reasons are listed.

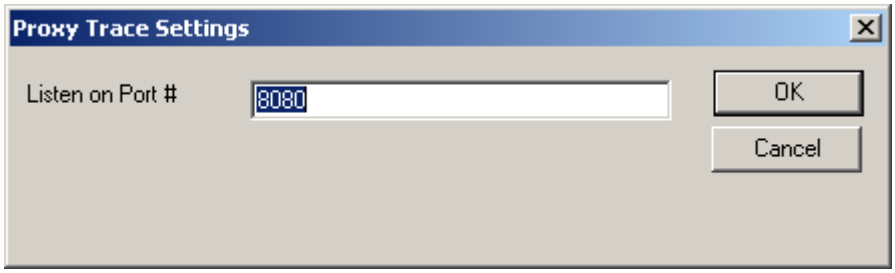
| Symptoms | Corrective Actions |
|---|---|
| The start menu item “Kofax Communication Server \| Configure TCOSS Web Services” on the TWS server is not available or not working. | (Re-)install TWS from the latest KCS release |
| TWS fails to start with event ID 9500. The description contains: “bind to port 80 failed” | Any other application already uses the TCP/IP port that is configured for TWS<br>• Change the used port in TWS<br>• Terminate the other application (e.g. Web server) that uses the TWS port. |
| TWS Web portal is not available | • Use TCMON to check if TWS is running<br>• Check if the URL in the web browser is correct<br>• Open the TWS configuration and check the configured port and https checkbox. |
| When I try to call the function “tcsi read sever time” in the web page responds with “612 server temporarily busy“ | • Open “Configure TCOSS Web Services” on the TWS computer and check if the TCOSS Server Path is correctly configured<br>• Restart TWS to ensure that the current configuration is valid.<br>• Use any other KCS client (e.g. TCfW or License Tool) to check if the connection to TCOSS is possible |
| When I try to call the function “tcsi read sever time” in the web page responds with “621 Registration Limit reached“ | There is no free TWS license on the TCOSS Server<br>• Use the KCS License tool to check if you have enough TWS licenses on the TCOSS server. |

| Symptoms | Corrective Actions |
|---|---|
| When I try to call the function “tcsi read sever time” in the web page responds with “610 wrong password“ | A wrong user ID/password is configured in TWS. There are 2 possibilities to solve the problem.<br>• Delete the user ID/password in the TWS configuration and restart TWS. Next time the browser will prompt for a user ID and password<br>• Use a correct TCOSS user ID/password in the TWS configuration. |

# HTTP Proxy Tracer

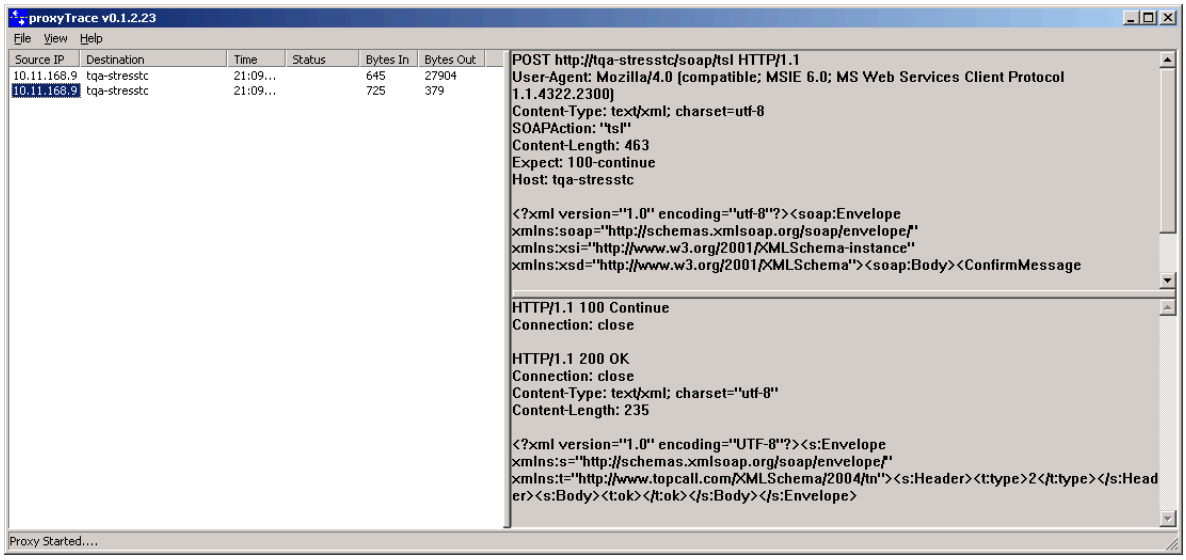
You can use any HTTP proxy tracer to capture the network traffic between you client application and TWS. A very simple freeware Proxy Tracer can be downloaded from the following address http://www.pocketsoap.com/tcptrace/pt.aspx.

After downloading the ZIP file you must extract the executable. It can be started from any directory. No installation is required. It will prompt you for the TCP/IP port that should be used. You can use any free port on your machine. Port 8080 is shown by default.




As next step, you must change your application to use the proxy traces as a proxy server. If the proxy server is running on your local machine, you must specify the IP address or host name of your computer. Do not use localhost, it will not work! You can test the proxy tracer with the ReceivePDF example described in chapter ReceivePDF. Look into the source code of this example to get more details how to call the web service via proxy server when using Visual Studio .net.

A screenshot of a simple web service call is shown below:


proxyTrace v0.1.2.23
File View Help
Source IP Destination Time Status Bytes In Bytes Out
10.11.168.9 tqa-stresstc 21:09... 645 27904
10.11.168.9 tqa-stresstc 21:09... 725 379
POST http://tqa-stresstc/soap/tsl HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; MS Web Services Client Protocol 1.1.4322.2300)
Content-Type: text/xml; charset=utf-8
SOAPAction: "tsl"
Content-Length: 463
Expect: 100-continue
Host: tqa-stresstc
<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"><soap:Body><ConfirmMessage
HTTP/1.1 100 Continue
Connection: close
HTTP/1.1 200 OK
Connection: close
Content-Type: text/xml; charset="utf-8"
Content-Length: 235
<?xml version="1.0" encoding="UTF-8"?><s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/" xmlns:t="http://www.topcall.com/XMLSchema/2004/tn"><s:Header><t:type>2</t:type></s:Header><s:Body><t:ok></t:ok></s:Body></s:Envelope>
Proxy Started....

Chapter 11

# Acknowledgements

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (http://www.openssl.org).

This product includes cryptographic software written by Eric A. Young (eay@cryptsoft.com).

This product includes software written by Tim J. Hudson (tjh@cryptsoft.com).

Chapter 12

# Conformance to Standards

The web service interface is based on the Simple Object Access Protocol (SOAP) 1.1, W3C Note of 8-May-2000 (see http://www.w3.org/TR/2000/NOTE-SOAP-20000508/)

The provided Web Service Description Language (WSDL) files use Version 1.1, W3C Note of 15-March-2001 (see http://www.w3.org/TR/wsdl)

**Note** All web service calls use the “document-literal” combination of SOAP binding style and data encoding. For a discussion of “document-literal” vs. “rpc-encoded” models, please see e.g. http://java.sun.com/developer/technicalArticles/xml/jaxrpcpatterns/,

http://msdn.microsoft.com/library/en-us/dnwebsrv/html/rpc_literal.asp

http://www-128.ibm.com/developerworks/webservices/library/ws-whichwsdl/

The HTTP layer implements HTTP/1.1 as defined in RFC 2616 (see http://www.w3.org/Protocols/rfc2616/rfc2616.html)

The HTTPS implementation is based on the OpenSSL Project, an Open Source toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols. OpenSSL version 0.9.8m from 25-Feb-2010 is used (see http://www.openssl.org).

The micro workflow component implements a subset of XSL Transformations (XSLT) Version 1.0 (see http://www.w3.org/TR/xslt).

Chapter 13

# Restrictions

- TWS does not use own Performance Counter or SNMP traps. But, event log entries generated by TWS can generate SNMP traps is with any other Kofax application be using TC/SNMP. Currently, TWS generates only event ID 9500. It contains the error reason if application has been terminated due to a failure in any component
- It is currently not supported to make an own Micro Workflow. If you have a demand for any new function please contact Kofax.
- There is no error message (e.g. event log entry) if the certificate will expire or is already expired. But the clients that access the server will get an appropriate warning.

Chapter 14

# Possible Future Enhancements

Configuration option for maximum number of concurrent web service calls. This value is currently fixed to 10.

Configuration option for the timeout after which clients are disconnected.

Show the native SOAP requests and responses on the Test server, similar as ASP.NET Web services do it.

Redirect to Web portal if default web page (without files/index.html) is called.

Support include/exclude IP range (which clients are allowed to use the Web service)

Changing the configuration settings from any remote web browser (this requires additionally some authentication in TWS to ensure that only privileged persons can change the configuration).

Chapter 15

# TWS Glossary

This section contains abbreviations and glossary.

## Abbreviations

| | |
|---|---|
| HTTP | Hyper Text Transport Protocol |
| PDF | Portable Document Format |
| SOAP | Simple Object Access Protocol |
| TCSI | KCS Client Server Interface |
| UCS | Universal Multiple-Octet Coded Character Set |
| UTF-8 | UCS Transformation Format 8 |
| WSDL | Web Services Definition Language |
| XML | Extensible Markup Language |
| TCXL | KCS XML Link Format |

## General Terms

**Extensible Markup Language (XML)**
An open standard for encoding structured data.

**Hyper Text Transport Protocol (HTTP)**
A transport layer that is used by web browser and web services.

**Impersonation**
Impersonation is used if an application gets the permissions of the acting user.

**Portable Document Format (PDF)**
A file format that has been defined by Adobe.

**Simple Object Access Protocol (SOAP)**
Alternative name for the Web service.

**UCS Transformation Format 8 (UTF-8)**
A multi-byte character code that is able represent all Unicode characters.

**User Credential**

Synonym for information provided to identify a user. Typically, the user credential consists of user id and password.

**Web Service**

An open standard for using functions of a server. The request and response data will be encoded with XML and transported on an http/https connection.

**Web Services Definition Language (WSDL)**

An open standard describing the functions of a web service interface.

**XPath**

An open standard describing an expression that returns data from an XML document.

**XSLT**

A language for transforming XML documents.

# KCS-Specific Terms

**Micro Workflow Component**

Component implementing a simple workflow. The workflow description is defined in the configuration using a subset of standard XSLT transformations with KCS specific extensions.

**TNC Streaming XSLT Engine**

TNC implementing a subset of standard XSLT transformations. But as an additional feature, the transformation is done by streaming which means that there is no message size restriction.

**TNC TCOSS 7 Connector**

TNC interfacing with Kofax messaging (TCOSS 7) or Archive Server. It gets requests from the TN and forwards it (with some transformations) to TCOSS 7 or Archive Server.

**TNC Web Service Connector**

TNC implementing the server side (and maybe in the future the client side) of a standard web service. It gets Web Service requests via http/https and forwards it to any other TNC (according to its configuration).

**KCS Client Server Interface (TCSI)**

Proprietary interface between TCOSS 7 server and clients

**KCS XML Link Format (TCXL)**

TCXL is a straight forward mapping of TCSI objects between XML syntax. The TCXL format describes all features supported by the TCOSS of TC/Archive server.