SMASER

TRIP Connectivity Framework

Operating Manual

Version 1.5
UNIX and Windows

Copyright © 2024 Smaser AG

End User License Agreement

All rights to this software, its documentation and logotypes of the TRIP product family and software (altogether “Software”) supplied by Smaser AG (Smaser) are exclusively owned by Smaser.

The transfer of this Software, solutions or parts thereof requires the prior written agreement of Smaser. Furthermore, the customer has the right to use licensed Software and / or process solutions supplied by Smaser to the extent specified in his contract with Smaser.

The free-to-use non-commercial version doesn’t require a prior written agreement with Smaser but such customers, organizations and/or third parties agree by using the software and / or solution of Smaser to be strongly obliged to keep all rights to this software, documentation and logotypes of the TRIP product family absolutely uninfringed and protected.

Table of Contents

Copyright © 2024 Smaser AG

Introduction

About this Document

The TRIP Connectivity Framework (TRIPcof) provides access to file filter technologies and to data import connectors. File filters provide text extraction and HTML conversion components for use with TRIPsystem and selected TRIP SDK products. Import connectors are used to import data from other data sources into TRIP for the purpose of search and/or data reuse.

The collection of supported file formats and connectors is not a static set. Extensibility is at the heart of the product and custom file format filter adapters and import connectors can be written in Java and .NET to support file formats and data sources not handled by the base product.

This manual for the TRIP Connectivity Framework describes:

- Common use cases and deployment scenarios
- How to use TRIPcof to add text extraction and HTML conversion features to applications
- How to develop custom file filter adapters for use with TRIPcof.

Related Documents

Installation procedures and configuration options are not covered by this manual. Please refer to the TRIPcof Installation Guide for more information on this topic.

Application development using TRIPnxp and TRIPjxp is not covered by this manual. Please refer to the reference manuals for respective product and the "TRIPnxp & TRIPjxp Programmer's Guide."

Contents of Product

The current version of TRIPcof contains the following components:

- Text extract filter for server execution (Windows, Linux, Solaris) and for client-side use from TRIPnxp and TRIPjxp.
- HTML conversion filter for server execution (Windows, Linux, Solaris) and for client-side use from TRIPnxp and TRIPjxp.
- SDK for Microsoft.NET and Java for the development of custom file filter adapters.
- Import connector framework that includes an SDK for Microsoft.NET for the development of custom import connectors.
- Import connector for file systems.

Differences from TRIPview

TRIPcof is designed to be compatible with TRIPview (and TRIPview-C) with respect to the text extraction and HTML conversion APIs in TRIPnxp and TRIPjxp. However, TRIPcof is not TRIPview and its behavior is not the same in all aspects. TRIP applications should therefore be validated against TRIPcof before TRIPcof is taken into production replacing TRIPview or TRIPview-C. Below is a description of differences and known possible issues.

- **No TRIPviewer**. This TRIPview component is based on an OEM product included in TRIPview and not in TRIPcof. It also requires TRIPclient, an older technology

which no longer is recommended for use to create Windows based TRIP applications. This technology is therefore not available in TRIPcof.

- **TRIPclient use** is not supported. The TRIPview APIs in TRIPclient are not compatible with TRIPcof. In this situation we recommend to continue using TRIPview or to rewrite such applications in TRIPnxp.
- **TRIPjtk use** is not supported. The TRIPview APIs in TRIPjtk are not compatible with TRIPcof. In this situation we recommend to continue using TRIPview or to rewrite such applications in TRIPjxp.
- The quality and availability of **HTML hit highlighting** depends on the file filter adapter used for HTML conversion. The ability to convert a file to HTML does not automatically mean that hit highlighting is supported, nor that the highlighting exactly matches the hit locations in TRIP search results. Please refer to the sections on the individual file filter adapters further on in this document for more detail.
- Extraction of **document properties** during text extraction depends on whether the file filter adapter and its underlying technology support property extraction.
- The structure and quality of the **extracted text** depends on the file filter adapter used. Different adapters use different underlying technologies to parse files. Two adapters who support the same file format may therefore produce a slightly different structure of the extracted text.

Differences from TRIPagent

The import connector functionality in TRIPcof addresses the same type of use cases that the now long-discontinued enterprise search module TRIPagent did. While conceptually quite similar, they differ in some aspects:

- Connectors run on the TRIP server, bypassing the overhead of the TRIP network communications protocol that caused TRIPagent to take a significant performance hit. Talking directly to the TRIP database kernel allows TRIPcof to operate at maximum efficiency with regard to reading and writing TRIP databases.
- The agent API in TRIPagent was more of an afterthought, as it was not originally designed to be a public part of the product. It was therefore not always easy or intuitive to use. TRIPcof comes with APIs for .NET and Java that simplifies the process of writing custom connectors, allowing developers to focus more on the data to import than on the use of the connector API.

Use Cases and Deployment

File Filter Use Cases

Indexing: Text extraction without storage

Brief description This use case applies to you if you want to make your existing file-based information searchable in TRIP, but do not wish to store the actual files in TRIP. TRIP will primarily be used as a search engine.

Benefits The TRIP database (BAF file) will be significantly smaller than if document copies are also stored in TRIP.

Disadvantages	Conversion to HTML will not be possible, since this requires that the file to be converted is stored in TRIP.
Application considerations	Consider using direct client-side text extraction to boost performance. This means that TRIPcof must be installed on the same machine with the (TRIPjxp/TRIPnxp based) application. If TRIPcof is used in server-side operation together with TRIPsystem, please note that there will be some extra overhead caused by transmitting each file to process across the network to the server.

Archiving: Text extraction and storage

Brief description	This use case applies to those who want to create an archive or content management style system that stores copies of the original documents (or the actual original documents) in TRIP. TRIP will be used as both a database and a search engine.
Benefits	The full range of TRIPcof functionality is available, including HTML conversion.
Disadvantages	The size of the TRIP database (BAF file) will be much larger compared to a database in which only the textual content of the documents is stored.
Application considerations	Only use server-side operation for text extraction. This minimizes the amount of data that has to be sent between the application and the TRIP server.

Publishing: Storage and HTML conversion

Brief description	This use case applies if you want to publish documents stored in a TRIP database to end users via the web, but only provide an HTML rendition instead of a copy of the original document.
Benefits	Returning an HTML version of a document can be a good idea if you want to keep the original to yourself or if the file format of the document is unusual and therefore unlikely to be readable by all users.
Disadvantages	HTML renditions are not perfect replicas. They will at best be an approximation. Some files can take a significant time to convert to HTML. Especially files that contain a lot of images or files that are image coded as a whole (such as certain types of PDF).
Application considerations	Direct client-side operation is recommended, because a generated HTML file is often accompanied by a set of image files. Transferring these from the TRIP server to the application may be complicated. Instead, fetch the document to convert to the application, then have the application call TRIPcof directly (via TRIPnxp or TRIPjxp) to perform the conversion. **If Multipart MIME (a.k.a. Web Archives, or .MHT files) is used**

as output from the HTML converter, please note that not all browsers will be able to display this format. You will need a browser extension to be able to view this format other browsers than Internet Explorer. The benefit of the Multipart MIME format is that HTML conversion can be done without adverse problems even with only a server-side installation of TRIPcof.

If large files (e.g. PowerPoints with over a hundred pages) are converted to HTML using server-side operation, be prepared that it may take some time. The default timeouts for a single c/s request are all set to one minute. Consider raising these timeouts (in TRIPnxp/TRIPjxp as well as in TRIPcof).

Import Connector Use Cases

Use TRIP as a search engine

Brief description Since TRIP has powerful search capabilities, it can be appealing to regard TRIP more like a search engine than a database. This use case involves importing data from other data sources into TRIP so that TRIP can be used to search in those data sets. The imported data is used for search only.

Benefits TRIP will be used to search in data imported from multiple external data sources, some of which may not have good search capabilities themselves.

Disadvantages TRIP is not a pure search engine. TRIP, also being a NoSQL database, requires data to be stored in order to be indexed. Duplication of some data is therefore unavoidable.

Application considerations TRIPcof will maintain one or more databases containing imported data, making sure that it is regularly synchronized with changes to the original data.

The responsibility of applications is therefore only to provide end users search interfaces, possibly customized to the type of data imported.

Importing data for reuse

Brief description NoSQL databases such as TRIP can be used to aggregate data from other data sources so that the data can be further refined and/or more conveniently reused. This use case involves using TRIP as an aggregation database by keeping copies of extracted data, possibly for further refinement.

Benefits TRIP will be used as a database and search engine for potentially quite different types of data, and applications can work with all manners of data using only TRIP.

Disadvantages Aggregation of data always some degree of redundancy, and this use case can effectively involve the creation of a parallel version

	of the data that possibly may diverge from the original.
Application considerations	TRIPcof is used to import data from other data sources. Once imported, data is treated as a new, parallel version. Synchronization of the imported data with changes to the original data is typically not done. Applications will likely be more than simple search portals, and also used to manipulate and refine the data in various ways.

Deployment Options

TRIPcof can be installed on the server-side together with TRIPsystem, or on the client-side together with TRIPnxp or TRIPjxp. The catch-all installation is the server-side one, but a client-side installation can sometimes be preferable or used as a complement.

Server-side

Install TRIPcof together with TRIPsystem especially if:

- You want to store copies of the documents you extract text from in TRIP.
- You want to be able to use TRIPcof from any TRIP SDK product.
- You want to use the import connector functionality in TRIPcof.

Client-side

Install TRIPcof together with TRIPjxp or TRIPnxp especially if:

- You do not want to store copies of the documents you extract text from in TRIP.
- You want to perform HTML conversion to plain HTML.

TRIPview Considerations

Although it has been several years since the last and final version of TRIPview was released, it supports a larger number of file formats for text extraction and HTML conversion. TRIPview can therefore be considered to be more powerful than TRIPcof when such operations are concerned.

This means than when using the text extraction and HTML conversion APIs in TRIPnxp and TRIPjxp, they will always attempt to use TRIPview if it is installed. If you wish to ensure that TRIPcof is used instead, you must not have TRIPview installed.

File Conversion

File conversion refers to the change of the representation of the file content from one format to another. Conversion to plain text is referred to as "text extraction".

TRIPcof currently supports plain text and HTML as target formats for conversion.

Text Extraction

Overview

The process of extracting text from a file is almost identical when using TRIPcof and TRIPview. The same classes and methods are involved in TRIPjxp and TRIPnxp.

TRIPcof also utilizes a backward compatibility layer that attempts to assure that also the APIs in TRIPnxp and TRIPjxp that were originally written for TRIPview also can be used

with TRIPcof, although the lack of filename as input argument in some of the backward-compatible APIs means that TRIPcof is less likely to be able to locate a suitable file filter adapter to use to extract text. Both TRIPnxp and TRIPjxp have from version 7.2 APIs that are fully TRIPcof compliant.

Prerequisites
If you wish to use TRIPnxp or TRIPjxp with TRIPcof, you must avoid using the `TdbStringField.ExtractionTarget` property. This has been made obsolete in favor of the `TdbTextField.TextExtractionInfo` property. Continuing use of the TdbStringField.ExtractionTarget property is not guaranteed to produce satisfactory results.

Server-side text extraction using TRIPjxp or TRIPnxp
Text extraction using TRIPjxp or TRIPnxp involves the use of the `TdbTextExtractionInfo` class via the `TdbStringField.ExtractionTarget` property.

C# snippet, adapted from the CofClient sample in TRIPnxp, located in the directory `samples\CofClient` under the TRIPnxp installation (for the Java equivalent, please refer to the CofExtract example):

```
001 : TdbTextField fText =
       (TdbTextField)head.CreateField(db.GetFieldByName("FILE_TEXT"));

002 : fText.TextExtractionInfo.Stream =
       new System.IO.FileStream(filename, FileMode.Open);

003 : fText.TextExtractionInfo.FileName = filename;
004 : fText.TextExtractionInfo.BinaryCopyField = "FILE_BLOB";
005 : fText.TextExtractionInfo.PropertyNameField = "PROP_NAME";
006 : fText.TextExtractionInfo.PropertyValueField = "PROP_VALUE";
007 : fText.TextExtractionInfo.ExtractText = true;
```

This code does the following, line by line:

1. Obtains a field object for the FILE_TEXT field that is to receive the extracted text from a file
2. Opens a stream to the file data and assigns it
3. Assigns the name of the file (note: required argument for TRIPcof)
4. We want to store a copy of the file, so we specify FILE_BLOB as the field to receive a copy of the file data
5. We want to extract document properties, so we specify PROP_NAME as the field to hold the property names
6. We want to extract document properties, so we specify PROP_VALUE as the field to hold the property values - this field is tupled with PROP_NAME
7. Set the ExtractText property to true to tell TRIP perform the text extraction upon the commit of the record to which this field object is associated

For more information about text extraction in TRIPjxp and TRIPnxp, please refer to the TRIPnxp & TRIPjxp Programmer's Guide version 2.2 or later, section "Interaction with TRIPview and TRIPcof".

Client-side text extraction using TRIPjxp or TRIPnxp
Client-side text extraction requires a local installation of TRIPcof on the same machine where TRIPjxp or TRIPnxp is running. The procedure is the same as for Server-side text extraction above, aside from the addition of a single property assignment:

```
fText.TextExtractionInfo.ClientSide = true;
```

This tells TRIPnxp / TRIPjxp to find the local installation of TRIPcof (or TRIPview-C instead if that is installed locally) and use it directly instead of going via TRIPsystem.

- TRIPnxp will look up the TRIPcof installation location TRIPCOF_HOME from the registry.
- TRIPjxp requires TRIPCOF_HOME as system property or as environment variable assigned the fully qualified path to the TRIPcof installation directory. If not assigned, TRIPjxp cannot find TRIPcof and the text extraction procedure will fail.

Alternatives for text extraction
Text extraction always involves storing the extracted text into a text field. The following combinations exist in which this may be performed.

- Uploading the contents of a file for text extraction, storing only the extracted text.
- Uploading the contents of a file for text extraction, storing the extracted text as well as a copy of the original document. This alternative enables the use of HTML conversion of the stored file.
- Storing the raw binary data of a file into a STRING field and later, possibly in another session, extract the text of from the file data stored in this STRING field.

In addition, you can choose to store document properties discovered during text extraction into a tupled pair of PHRASE fields, where one field holds the property names and the other holds the property values.

Text extraction can be done server-side (using TRIPcof as a plug-in to TRIPsystem) or client-side (using a local TRIPcof installation directly from TRIPnxp or TRIPjxp).

HTML Conversion

When using HTML conversion with hit highlighting, the highlighting information will be taken from a TEXT field that contains the extracted text of the document. This field MUST be defined with retained layout ("ORIG") in order for highlighting offsets to be correct.

Overview
The process of converting a document to HTML is almost identical when using TRIPcof and TRIPview. The same classes and methods are involved in TRIPjxp and TRIPnxp.

Prerequisites
If you use TRIPnxp or TRIPjxp with TRIPcof, you must:

- For server-side HTML-conversion, you must use the classes `TdbRendition` or `TdbHighlightRendition` with the constructor that takes three parameters and as second parameter takes the name of a PHRASE field in which the original name of the file to convert is stored.
- For client-side conversion, use the method `Convert` on the `TdbStringField` class.

- If you wish to use highlighting, ensure that you have a file filter adapter installed capable of highlighting the results and that it is enabled.

Server-side HTML conversion using TRIPjxp or TRIPnxp
Server-side HTML conversion using TRIPjxp or TRIPnxp involves the use of retrieval templates. Use `TdbRendition` for normal output and `TdbHighlightRendition` for highlighted output.

Java snippet, adapted and simplified from the CofConvert sample in TRIPjxp (for the .NET equivalent, please refer to the CofClient sample in TRIPnxp):

```
001 : TdbRecord tmpl = new TdbRecord(session);
002 : tmpl.addToTemplate(new TdbHighlightRendition("FILE_BLOB","FILE_TEXT",
          "FILE_NAME", TdbRenditionType.MimeEncodedHtml));

003 : TdbSearch s = new TdbSearch(session);
004 : s.setRetrievalTemplate(tmpl);
005 : s.setAutomaticRetrievalTemplate(false);

006 : s.execute("BASE cofdemo");
007 : s.execute("FIND trip#");
008 : TdbSearchSet ss = s.getLastSearchSet();

009 : TdbRecord rec = ss.getRecord(0);
010 : TdbStringField str = (TdbStringField) rec.getHead().getField("FILE_BLOB");
011 : str.copyToFile(outputDirectory + "mydocument.mht");

012 : s.close();
```

This code does the following, line by line:

1. Creates a TdbRecord instance to act as a retrieval template.
2. Creates an instance of TdbHighlightRendition for HTML output as highlighted multipart MIME and adds this to the retrieval template.
3. Creates a new TdbSearch instance.
4. Assigns the retrieval template to use
5. Disables creation of automatic retrieval templates
6. Opens the database in which the document to convert is stored.
7. Executes a query to locate the document
8. Obtains the TdbSearchSet instance that represent the latest search.
9. Get the first record from the search result.
10. Get the field object for the returned converted value of the FILE_BLOB field.
11. Write the conversion result to a file.
12. Close the TdbSearch instance and delete its associated search sets.

For more information about HTML conversion in TRIPjxp and TRIPnxp, please refer to the TRIPnxp & TRIPjxp Programmer's Guide version 2.2 or later, section "Interaction with TRIPview and TRIPcof".

Client-side HTML conversion using TRIPjxp or TRIPnxp
Client-side HTML conversion extraction requires a local installation of TRIPcof. Note that TRIPview-C it will be used instead if it is installed.

The procedure involves the use of the method Convert of the TdbStringField class. This method is dedicated to client-side conversion and cannot be used for server-side operation.

C# snippet based on the CofClient sample in TRIPnxp (for the Java equivalent, please refer to the CofConvert example in TRIPjxp):

```
001 : TdbRecord tmpl = new TdbRecord(session);
002 : tmpl.AddToTemplate("FILE_BLOB");
003 : tmpl.AddToTemplate("FILE_NAME");
004 : tmpl.AddToTemplate("FILE_TEXT");

005 : TdbSearch s = new TdbSearch(session);
006 : s.RetrievalTemplate = tmpl;
007 : s.AutomaticRetrievalTemplate = false;

008 : s.Execute("BASE cofdemo");
009 : s.Execute("FIND trip#");
010 : TdbSearchSet ss = s.LastSearchSet;
011 : TdbRecord rec = ss[0];

012 : TdbComponent h = r.Head;
013 : TdbStringField blob = (TdbStringField) h.GetField("FILE_BLOB");
014 : TdbPhraseField name = (TdbPhraseField) h.GetField("FILE_NAME");
015 : TdbTextField   text = (TdbStringField) h.GetField("FILE_TEXT");

016 : String storedName = "";
017 : if (name.ValueCount > 0)  storedName = name.Values[0].Trim();
018 : if (storedName.Length == 0) storedName = "dummy.doc";

019 : blob.Convert(TdbRenditionType.MimeEncodedHTML, storedName,
         outputDirectory, "myDocument.mht", "apachepoi", text, true);

020 : s.close();
```

This code does the following, line by line:

1. Creates a TdbRecord instance to use as field template
2. Adds the field FILE_BLOB to the retrieval template.
3. Add the field FILE_NAME to the retrieval template.
4. Add the field FILE_TEXT to the retrieval template. Fetching the field in which the extracted text is stored is necessary if hit highlighting is to be applied. Can be omitted otherwise.
5. Create a new TdbSearch object.
6. Assigns the retrieval template to use
7. Disables creation of automatic retrieval templates
8. Opens the database in which the document to convert is stored.
9. Executes a query to locate the document
10. Obtains the TdbSearchSet instance that represent the latest search.

11. Get the first record from the search result.
12. Get the head component of the record
13. Get the field object for the FILE_BLOB field
14. Get the field object for the FILE_NAME field
15. Get the field object for the FILE_TEXT field
16. Declare a variable to hold the name of the original document.
17. If the field FILE_NAME contains a value, assign that to the file name variable.
18. If the original file name is not known assign a dummy value. This is necessary because TRIPcof requires the name of the original file even if the underlying file filter technology may not always do so.
19. Perform the conversion to highlighted HTML in multipart MIME format using a local TRIPcof installation.
20. Close the TdbSearch instance and delete its associated search sets

For more information about HTML conversion in TRIPjxp and TRIPnxp, please refer to the TRIPnxp & TRIPjxp Programmer's Guide version 2.2 or later, section "Interaction with TRIPview and TRIPcof".

Alternatives for HTML conversion
HTML conversion always involves processing file data stored in a STRING field. The only significant alternatives you have at your disposal are:

- Server-side or client-side operation?
- Hit highlighting or not?

Server-side HTML conversion utilizes TRIPcof as a plug-in to TRIPsystem and can be performed from applications based on TRIPjxp and TRIPnxp.

Client-side HTML conversion from TRIPnxp or TRIPjxp calls a local TRIPcof installation directly.

The important thing to remember with hit highlighting and TRIPcof is that different underlying file filter technologies may support different ways to apply such markup. This means that the highlighting may not always match the hit locations that TRIPsystem produces. If exact hit locations cannot be used with a particular technology, TRIPcof will instead highlight all words identical to those that hit.

Import Connectors

Import connectors are plugins to TRIPcof that are used to import data from external data sources into TRIP. Using the connector API, custom connectors can be written, allowing data from any kind of external data source to be imported into TRIP for search and/or data reuse purposes.

Configuring the standard connectors

For information about the configuration of the main configuration file for the import connector functionality in TRIPcof (cfw.conf), refer to the TRIPcof Installation Guide.

Configuring the file system connector (fsbot)

The “fsbot” file system connector is capable of indexing new and modified files, detecting deleted and renamed files, and can be run in the background in so-called active monitoring mode to detect changes in file system as they occur.

The configuration file for the “fsbot” connector is named “fsbot.conf” and is by default located in the TRIPcof installation under conf/cfw.connectors.d. This directory can be customized in the main connector configuration file cfw.conf (see TRIPcof Installation Guide).

Most of the properties in the fsbot.conf file must be left unchanged in order for the connector to operate correctly. The following properties may however be modified:

LogDir	The directory where log files are written. Default: ../../log
LogPrefix	A string that log file names will start with. Default: icn_fsbot
LogLevel	Names the log level that the connector will use. Valid values are: 0 Off 1 Fatal errors only 2 Errors 3 Errors and warnings 4 Errors, warnings and information 5 Errors, warnings, information and debug statements Default: 3

Setting up a data source for a connector

Connectors operate on data sources. What a data source is depends on the connector. For the “fsbot” file system connector, the data source is any directory tree on the local file system. For a hypothetical web connector, a data source would be a web site. For a custom connector that reads a data from a relational database, the data source could be a table or a view. For a hypothetical email connector, the data source could be an email account or even all email accounts on an email server. Et cetera.

Regardless of what kind of connector and data source to use, data sources are always configured in data-source-dedicated files in the conf/cfw.importds.d directory under the TRIPcof installation. What goes into a data source configuration file depends totally on the type of the connector, as they are quite likely to need rather different configuration properties.

Use the provided mkdscfg utility program to generate a template for a data source configuration file that suits the connector you want to use. A sample starter configuration file for the fsbot connector is also provided and can be found in the conf/samples.cfw directory under the TRIPcof installation.

Some of the data source configuration properties are common to all data sources, regardless of type and connector. These are:

[datasourcename]	The first line in the configuration must be the unique name of the data source within square brackets. A good practice is to prefix the name with the name of the connector. Replace the sample name with your own data source name. E.g. “fsbot-localfiles” for a file system data source.
Type	Identifies the section type. For data sources, this value must be “DataSource”. Default: DataSource (**do not change this value**)
Connector	The name of the connector to use with this data source. For file system data sources, this property must be set to “fsbot”. For custom data sources, set this property to the name of your custom connector.
Enable	Set to “True” to enable the data source and to “False” to disable the data source. Default: True
LocalStore	The name of the TRIPcof import connector database in which the imported data will be stored. Compatible databases can be created via TRIPmanager by selecting the “connector” database type, and similarly in a programmatic fashion via TRIPjxp and TRIPnxp. Default: TRIPCOF
PropertyMapFile	The name of the file containing the property map to use. This file contains a list of mappings between property names and field names, such that properties of the specified types go into the specified TRIP fields. The default name std_propmap.conf is used if no value is specified for this property.
UserMapFile	The name of a file that contains mappings of user and group names between the operating system and TRIP. The default name is tripcof.usermap, assumed to be located in the conf directory. To use the default file, this property can be left out. To disable mapping, set this property to an empty value. To use a custom mapping file (recommended if user/group mapping is used), specify the fully qualified path to the file here.
StoreCopy	Indicates if copies of indexed documents are to be stored in TRIP. Set to True to store a copy of the file in TRIP, and False to only store the extracted text (if extraction is possible). Default: False

Mapping document properties to TRIP fields
If the connector produces data that is accompanied by properties or similar meta data information, such meta data are normally stored in two tupled fields in TRIP; one field for the property names and the other for property values. This information can be searched using AND.T and KVP-style DISPLAY orders, and is a convenient way to represent any kind of property for an item. The connector database design uses the part record fields V_PROPNAME and V_PROPVALUE for this purpose.

In some circumstances it may be preferable to have properties assigned to their specific fields instead, so that author information goes into an author field, keyword information into a keyword field, etc. A default mapping is provided for common document properties that describe author, title, subject and description. To extend or modify this mapping, follow these steps:

1. Examine the std_properties.conf file. This is a configuration file that declares common names for document properties. This is needed because different file formats and sometimes different versions of the same format will contain property quite different names for the same property.

   If you extend this file, please note that any extensions will have to be reapplied after an upgrade of TRIPcof.

2. Implement or reuse a document property map file. This file maps property names as declared in std_properties.conf to TRIP field names. The default property map file std_propmap.conf is provided. If you wish to extend or modify this configuration, you should create a copy, then edit the copy. Do not edit the file itself.

3. Add a PropertyMapFile line to your data source configuration file(s) that you wish to use property mapping with. The value is either a relative or absolute path to the property map file you selected in step 2.

Mapping OS user and group names to TRIP
The data imported by a connector may have specific user access privileges associated with each item. A file from a file system, for example, will typically have an access control list specifying users and groups that have (or shouldn't have) access. This information is imported into TRIP as is, but the operating system users and group names may not always correspond to TRIP user and group names. To make sure that the OS user and group names are also inserted as their TRIP counterparts, the data source configuration must include a UserMapFile property.

The value of the UserMapFile property can be set in one of the following ways:

- Left out entirely or commented out. This causes TRIPcof to use the default user map file (tripcof.usermap) located in the conf directory.

- Set to an empty value. This disables user and group name mapping for the data source.

- Set to the fully qualified path of the mapping file to use, preferably located outside the TRIPcof installation directory to make upgrades easier.

Creating a file system data source (fsbot)
The simplest way to create a data source for the fsbot connector is to copy the fsbot.sampleds.conf file to the cfw.importds.d directory and give it a new name. Open the copy in a text editor to customize it. The table below list the fsbot-specific properties:

StartPath	The name of the directory at the root of the directory tree to index. To refer to a remote directory (share) on WIndows, use a UNC path (e.g. `\\server\share\my\directory`).
Mount	Indicates if the StartPath indicates a mount point or a remote file system of some kind that need to be mounted by the connector. Set to True if the StartPath requires mounting, and False if it is a regular, local directory, or one that does not require explicit mounting. **Windows**: If not set, this property will default to True if the StartPath is a UNC path, and False otherwise. **Unix/Linux**: If not set, this property will default to False. **NB**: *Mounting is currently only supported on Windows.*
FileShareUser	**Windows only**: Username including domain for the user to be used to to mount the file share referred to by the StartPath. This property is only considered if StartPath is a UNC path. **NB**: Leaving the FileShareUser and FileSharePass properties unset is possible if the user that runs the connector process has rights to mount the remote directory indicated by StartPath.
FileSharePass	**Windows only**: Password for the user indicated by the FileShareUser property.
Excludes	A semicolon-delimited list of patterns for names of file or directories that must be ignored. The * wildcard is supported.
Includes	A semicolon-delimited list of patterns for names of files that must be processed. The * wildcard is supported. If undefined or empty, all files except those indicated by the Excludes property will be processed. If defined, only the files that match the patterns specified here will be processed.

NOTE

The “fsbot” file system connector may not be able to:

- Obtain access control information for files read from a network file share.
- React to change events on a network file share. Running a connector in active

monitoring mode against a network file share is therefore not supported.

If access control information is required and/or active monitoring use is required, the connector should run on the file server itself.

The mkdscfg utility program can be used to generate a template for a configuration file suitable for the fsbot connector. For example:

```
mkdscfg -i fsbot -n MyFileDataSource -o myfileds.conf
```

The complete list of arguments and options for the mkdscfg tool can be seen if it is started using the “--help” argument.

Using import connectors

The use of an import connector to retrieve data from a data source for storage and indexing in TRIP involves running the cfwimport command line tool. This tool can be used in the following ways:

- Manual interactive use to index new and/or modified data, or to check for deleted data.
- Scheduled invocation (via separate third-party scheduling software) to perform regular indexing of new and/or modified data, and to check for deleted data.
- Background process with active monitoring, indexing changes in the data source as they occur. This requires that the connector is written to support active monitoring. This is currently only the “fsbot” file system connector.

In all three cases, a specific data source must be selected. Processing multiple data sources at the same time is not supported, but must be done as separate calls to cfwimport.

NOTE

Even if the associated TRIP database is populated with new or modified data from the data source, it will **not be indexed automatically**. This is best practice for bulk loading data into TRIP. Indexing the TRIP database should therefore be done as a separate step after the cfwimport process successfully completes and no additional data will be imported for the time being.

The complete list of arguments and options for the cfwimport tool can be seen if it is started using the “--help” argument. For instance, on Windows:

```
USAGE: cfwimport [options]

Options:
  -conf       <file>  Read configuration from this config file
  -datasource <id>    The data source to connect to

  -list               List data sources
  -new                Locate new items
```

```
  -modified           Locate modified items
  -deleted            Clear out deleted items from index

  -active             Run in active (event triggered) mode
  -headless           Run in non-interactive mode
  -daemon             Run as a service (implies -active and -headless)
  -in         <file>  File to process (non-active/daemon use only)

  -loglevel   <level> Set logging level (0=off, 5=debug)
  -logdir     <dir>   Override logging directory

  -install            Install cfwimport as a service
  -uninstall          Uninstall the cfwimport as a service
  -username   <name>  DOMAIN/USERNAME for user to run the service as
  -password   <pwd>   The password for the specified user
```

Listing existing data sources
To list configured and enabled data sources for all connectors, use the “-list” option to the cfwimport program. The output will be a list of the data source names (as specified within the square brackets on the first line of the data source configuration file).

Example:

```
C:\> cfwimport -list

fsbot-local
myconnector-ds1
```

The above example shows that there are two data sources available for use; the “fsbot-local” and the “myconnector-ds1”.

Processing new and modified data in a data source
To import new and modified data in a data source, use the “-new” option to the cfwimport program. You may want to specify a logging level of at least 4 to get a bit more information on what is going on while the connector is working.

Example:

```
C:\> cfwimport -datasource fsbot-local -new -loglevel 4
```

Processing only modified data in a data source
To only check a data source for updates to already imported data, use the “-modified” option to the cfwimport program. You may want to specify a logging level of at least 4 to get a bit more information on what is going on while the connector is working.

Example:

```
C:\> cfwimport -datasource fsbot-local -modified -loglevel 4
```

Checking a data source for deleted data
To only check a data source for deletions of already imported data, use the “-deleted” option to the cfwimport program. You may want to specify a logging level of at least 4 to get a bit more information on what is going on while the connector is working.

Example:

```
C:\> cfwimport -datasource fsbot-local -deleted -loglevel 4
```

Running active monitoring interactively
When running in active monitoring a connector will detect changes in data source as they are happening. The standard “fsbot” file system connector supports this mode of operation.

To start cfwimport in active monitoring mode, use the “-active” option. You may want to specify a logging level of at least 4 to get a bit more information on what is going on while the connector is working.

Example:

```
C:\> cfwimport -datasource fsbot-local -active -loglevel 4
```

In this mode, the connector will run indefinitely, or until a critical error is detected. To stop the cfwimport program when it is running in active monitoring mode, press Ctrl-C on Windows and Ctrl-D on Linux.

To guarantee best results for active monitoring on Windows, the cfwimport program should be run as an administrator user with UAC elevated privileges.

Using cfwimport from scheduler software (e.g. Cron on Linux)
The only difference between manual interactive use of cfwimport and scheduled use is that when used from a scheduler software it gets more important to generate log files that can be examined in case something goes wrong. Otherwise all arguments and options remain the same.

By adding the “-headless” option to the cfwimport argument list, you inform it that it is running in non-interactive mode and cannot expect any kind of user interaction. This optional argument mainly affects console output, and can be safely omitted if console output happens to be desired also when executed from a scheduler.

Installing cfwimport as a service (Windows only)
To have a particular data source be actively monitored for changes all the time, the cfwimport program can be installed as a service for the data source in question. Multiple data sources can be run as services, each one will then get its own service name and configuration.

Example (run in a console opened in Administrative mode – “run as”):

```
C:\> cfwimport -datasource fsbot-local -install -loglevel 4
               -username MYDOMAIN\username
               -password password
```

The name of the service will be “cfwimport-“ followed by the name of the data source. The above example will result in the service “cfwimport-fsbot-local”. If creation of the service is successful, it will be configured to start automatically, but not started. Start the service in a regular fashion via the control panel or by using the “net start” command in a console started with administrative privileges (“run as”).

To uninstall a cfwimport service, use the “-uninstall” option:

```
C:\> cfwimport -datasource fsbot-local -uninstall
```

If “-uninstall” is specified without a data source, all installed cfwimport services will be uninstalled.

Using data imported by a connector

TRIPcof use the same, generalized database design for all connectors. Databases in this design can be created via TRIPmanager using the “connector” database type..

The principles behind its design are:

- Main focus on text-oriented data
- A record can represent a data item that have several parts (e.g. an email with attachments).
- Each data item part can have its own properties.
- Each data item part can have its own extracted text.
- Support for access control list entries (for data sources that can provide them).

Searching for by contents (extracted text)

The field X_TEXT contains the extracted text.

For example:

```
FIND X_TEXT=example
```

There is also a field group “content” that allows searching in several textual fields; the extracted text, text provided directly from the data source, and the fields holding extracted properties for document title and description. For example:

```
FIND CONTENT=example
```

Listing extracted document properties and their values

The fields V_PROPNAME and V_PROPVALUE are tupled and contain a list of the document properties that could be obtained from the source data.

To enumerate all property names:

```
DISPLAY V_PROPNAME=#
```

To search for documents having a particular property:

```
FIND V_PROPNAME='AUTHOR'
```

To search for documents having a particular property value:

```
FIND V_PROPNAME='AUTHOR' AND.T V_PROPVALUE='John Doe'
```

Alternatively, a KVP definition can be used. This would make all values in the V_PROPNAME field searchable as if they would be field names:

```
DEFINE KVP=V_PROPNAME,V_PROPVALUE
FIND AUTHOR='John Doe'
```

A KVP definition can also be used with display orders to enumerate the values of a property:

```
DEFINE KVP=V_PROPNAME,V_PROPVALUE
DISPLAY AUTHOR=#
```

Note that once a KVP is defined, it will stay defined until the end of the session. Redefinition is not necessary unless different fields are to be used.

Defining read scopes
If access control list data has been supplied by the connector(s), the I_ACL_* fileds will have contents. These fields indicate which users and groups have (or doesn't have) read access to the record. These names are from the data source, and do not necessarily describe TRIP users and groups.

I_ACL_USERS	Users with read access to this item
I_ACL_GROUPS	Groups with read access to this item
I_ACL_NOUSERS	Users with denied read access to this item
I_ACL_NOGROUPS	Groups with denied read access to this item

If user group mapping is enabled for the data source, the TRIP-specific counterparts of the above fields are populated. If the user and group names imported from the data source are not TRIP names, these are the ones that should be used in read scopes in order to enable record-level security, reflecting that of the data source:

I_TACL_USERS	Users with read access to this item
I_TACL_GROUPS	Groups with read access to this item
I_TACL_NOUSERS	Users with denied read access to this item
I_TACL_NOGROUPS	Groups with denied read access to this item

To make proper use of this information in TRIP so that users can only search and retrieve data that they have access to, read scopes should be defined.

A catch-all condition for a read scope could be:

```
((I_ACL_USERS="" AND I_ACL_GROUPS="" AND I_ACL_NOUSERS="" AND
I_ACL_NOGROUPS="") OR I_TACL_USERS=USER() OR
I_TACL_GROUPS=GROUP()) NOT I_TACL_NOUSERS=USER() NOT
I_TACL_NOGROUPS=GROUP()
```

This would match records that:

- Do not have any ACL defined (and therefore should be readable by anyone with access to the database). The fields that should used for this are the ones whose values are from the data source (e.g. OS user and group names).
- Contains the current TRIP user named in I_TACL_USERS
- Contains any of the current user's TRIP groups in I_TACL_GROUPS
- Do not have the current TRIP user named in the I_TACL_NOUSERS field and do not have the any of the current user's TRIPP groups mentioned in I_TACL_NOGROUPS.

If deny-access ACLs are not used in your organization, the conditions on the I_TACL_NOUSERS and I_TACL_NOGROUPS fields can be omitted from the scope query, simplifying it somewhat. For example:

```
(I_ACL_USERS="" AND I_ACL_GROUPS="") OR I_TACL_USERS=USER() OR
I_TACL_GROUPS=GROUP()
```

Refer to the TRIPsystem documentation for information on how to define scopes.

Custom File Filter Adapters

Concept Overview

There may be file formats you wish to process that are not supported by any of the file filter technologies for which TRIPcof has adapters. In order to be able to process such file formats, TRIPcof comes with an API that allows the development of custom file filter adapters.

A custom file filter adapter is a shared library (Unix/Linux) or dynamically linked library (Windows) that interfaces TRIPcof with an underlying file filter technology. The API used to build these custom file filter adapters is the same one used to create the file filter adapters included in TRIPcof.

Custom file filter adapters can be written in Java and using the Microsoft .NET framework in languages such as C# or VB.NET.

Isolation

A file filter adapter is always loaded in a process dedicated for the conversion of a single file. There will not be concurrent threads in the same process doing multiple conversions in parallel. Each individual file conversion will take place in its own isolated process. The reason for this is to isolate TRIP from any instability in the file filter adapters and in whatever underlying third party technology they may be using. The overhead incurred by running the conversion in a separate process is compensated by increased application and system stability.

Note that the current version of TRIPcof never performs more than one conversion in a single process. This may change in future versions, which may implement features that require multiple sequential conversions within the same, dedicated process.

Considerations:

- Write custom file filter adapters so that it avoids doing unnecessary work during load and initialization of the adapter and the conversion process.
- The custom file filter adapter must be able to handle multiple sequential conversions in the same process without overusing or leaking system resources.

Implementing an Adapter

Writing an adapter in Java

Using Java to write a custom file filter adapter involves subclassing the `FilterAdapter` class and using its associated support classes and JNI libraries.

A good place to start is to use the sample "fifi_java_sample" as a reference. It implements a bare-bones text extraction adapter, which you can use as a "skeleton" upon which to base your own adapter.

Writing an adapter in .NET
Using a .NET language to write a custom file filter adapter involves subclassing the `FilterAdapter` base class and using its associated support classes.

A good place to start is to use the sample "InfinITServices.Trip.FileFilter.Example" as a reference. It implements a bare-bones text extraction adapter, which you can use as a "skeleton" upon which to base your own adapter.

Conversion Procedure

After an adapter has been loaded into the conversion process, the adapter sees the following workflow.

1. The adapter is asked to initialize.

   This involves a call to the `InitializeAdapter` *method.*

2. The adapter is asked if a file with a given name can be converted to a specified output format.

   This involves a call to the `CheckConversion` *method that should perform a check based of file name suffix. Return Yes if the adapter definitely can handle the conversion, Maybe if the adapter is likely to be able to handle the conversion, or No otherwise.*

3. If the Connectivity Framework decides to use the adapter, the adapter will receive a reference to the file contents and asked to prepare for conversion.

   This involves a call to the `OpenConversion` *method..*

4. If the conversion involves hit markup (e.g. for HTML conversions), the adapter will be supplied with a character-based list of length/offset pairs describing locations in the text that should be highlighted.

   If hit highlighting is requested, hit information is available via the `HitWords` *and* `HasHitWord` *properties of the FilterAdapter base class.*

5. The adapter is asked to perform the requested conversion.

   This involves a call to the `Convert` *method which must implement the main conversion routine for the adapter.*

   Adapters are expected to call the methods `AddConversionResult` *and* `AddExtractedProperty` *to provide the result of a text extraction to the connectivity framework.*

   When performing HTML conversion, all adapters are required to write the result to a file named by the second argument to the `Convert` *method.*

6. When the conversion is complete (regardless of success/fail status), the adapter is asked to close the conversion.

   The adapter receives a call to the `CloseConversion` *method. This function or method should perform any cleanup required after the prior call to the* `Convert` *method,*

7. If conversions of additional files are requested within the same process, repeat from step 2. Otherwise continue.

8. The adapter is unloaded when the converter process shuts down.

   Just prior to this, the adapter receives a call to the `UninitializeAdapter` *function so that the adapter may perform a controlled clean up and release of resources.*

Error Handling

Whenever an error occurs in your processing such that you cannot continue, you must throw a `FileFilterException` and call `setError`:

```
setError(FilterReturnCode.ConversionError, "Conversion failed because...");
throw new FileFilterException(FilterReturnCode.ConversionError,
       "Conversion failed because...");
```

An alternative to this is use the `throwError` method, which will properly register the error state and throw an exception:

```
throwError(FilterReturnCode.ConversionError, "Conversion failed because...");
```

Failure to call the `setError` method directly or indirectly as described above may result in TRIPcof getting incorrect information about the status of the current conversion operation. Logging and application feedback may also get confusing and contradictory.

Logging

The file filter adapter framework supports logging. Java and .NET based adapters can use the `logDebug` method declared on the `FilterAdapter` base class to output debug logging statements. Also, whenever the `setError` method gets called to register a processing failure, the error message will also be logged, provided that the log level is 2 (errors) or higher.

All logging statements, even ones used for debug purposes, will only be output if the log level of the logging system is higher or equal to the log level of the statement. This means that you can safely leave them in place even in production. The performance hit for checking the logging level for each statement is negligible.

Configuration File

All adapters need their own configuration file. This also applies to custom adapters. Please refer to the conf/samples.fifi directory for template configuration file. It has detailed comments for each setting, so please study it carefully.

An adapter written in C# may for instance have the following configuration:

```
[mydotnetadapter]
Enable=True
Priority=25
Label=My .NET Adapter
Manufacturer=Example Inc
ConvLib=..\..\bin\InfinITServices.Trip.FileFilter.ApiWrapper.dll
WorkDir=..\..\tmp
TargetFormats=text/plain
SupportsProperties=False
SupportsHighlighting=False
IsDotNet=True
DotNetAssembly=..\..\bin\mydotnetadapter.dll
DotNetClass=Example.TripCof.MyAdapter

# Logging options
LogLevel=3
LogDir=..\..\log
LogPrefix=fifi_dotnetsample
```

Custom import connectors in .NET

TRIPcof comes with a Microsoft.NET API for writing custom import connectors. This section covers how to create one, the relevant parts of the API, how to create configuration files for the connector and the connector's data sources, and finally deployment.

Writing a connector

The following points outline the procedure:

- Create a Visual Studio project for a .NET Framework class library
- Add a reference to InfinITServices.Trip.ConnectorFramework.dll, which you can find in the TRIPcof bin directory.
- Create class for your connector and make it inherit from the abstract base class ImportConnector.
- Implement all abstract methods: Connect, Disconnect, FirstItem, NextItem, CheckItem and LoadItem.
- Optionally override the methods InitializeConnector and UninitializeConnector.
- If your connector uses custom data source properties (which is likely), override the method DescribeProperty to make sure that data source configuration templates generated using the mkdscfg tool are complete.

Instead of starting all of the above from scratch, you can use the provided sample project "InfinITServices.Trip.ConnectorFramework.SampleImportConnector" to base your connector upon.

Crawler Procedure for new and modified data

After a connector has been loaded into the driver process for scanning for new and modified data (typically using the cfwimport program invoked with the "-new" option), the connector sees the following workflow.

1. The connector is loaded and the connector class is instantiated. Properties loaded from the connector's configuration are provided.

   To query these properties, the connector should use the methods `HasConnectorProperty` *and* `GetConnectorProperty`*.*

2. The connector is asked to initialize.

   This the connector receives a call to the `InitializeConnector` *method, which may optionally be overridden. This happens only once during the lifetime of the process.*

3. Data source-specific properties are provided to the connector. These have been read from the data source configuration file by the connectivity framework.

   To query these properties, the connector should use the methods `HasConnectorProperty` *and* `GetConnectorProperty`*.*

4. The connector is asked to connect to the data source whose configuration properties were provided in step 3.

   The connector receives a call to the `Connect` *method. If the 'active' parameter is True and the connector does not support active monitoring, it must throw an exception using the* `ThrowError` *method.*

5. The connector is asked to return an `ImportItem` for the first data item.

   The connector receives a call to the `FirstItem` *method. The argument is the object returned from the connector on the call to the* `Connect` *method. If no data could be found at all, the connector should return null.*

   The `ImportItem` *instance must only contain the id of the data item and either a timestamp for the last change to the data item, or a checksum value. Do not load the data for the item at this stage!*

6. If the `FirstItem` call returned null, skip to step 11.

7. Unless the data item has been previously retrieved and stored in TRIP and has not been updated in the data source since then, the connector is asked to load the item's actual data. If the data item has not changed, this step will be skipped.

   The connector receives a call to the `LoadItem` *method. The argument is the object returned from the connector on the call to the* `Connect` *method. The second argument is an* `ImportItem` *instance identical to the one that was returned from the connector in step 5.*

8. The connector is asked to return an `ImportItem` for the next data item.

   The connector receives a call to the `NextItem` *method. The argument is the object returned from the connector on the call to the* `Connect` *method. If no more data can be found, the connector should return null.*

The `ImportItem` instance must only contain the id of the data item and either a timestamp for the last change to the data item, or a checksum value. Do not load the data for the item at this stage!

9. Unless the data item has been previously retrieved and stored in TRIP and has not been updated in the data source since then, the connector is asked to load the item's actual data. If the data item has not changed, this step will be skipped.

   The connector receives a call to the `LoadItem` method. The argument is the object returned from the connector on the call to the `Connect` method. The second argument is an `ImportItem` instance identical to the one that was returned from the connector in step 8.

10. If last `NextItem` call did not return null, repeat from step 8.

11. The connector is asked to disconnect from the data source.

    The connector receives a call to the `Disconnect` method. The argument is the object returned from the connector on the call to the `Connect` method.

12. The connector is unloaded when the process shuts down.

    Just prior to this, the connector may receive a call to the `UninitializeConnector` method, which may optionally be overridden. This happens only once during the lifetime of the process, and after this has occurred, the adapter will no longer be used in the current process.

Crawler Procedure for modified or deleted data only

After a connector has been loaded into the driver process for scanning for modified and/or deleted data (typically using the cfwimport program invoked with the "-modified" or "-deleted" options), the connector sees the following workflow.

1. The connector is loaded and the connector class is instantiated. Properties loaded from the connector's configuration are provided.

   To query these properties, the connector should use the methods `HasConnectorProperty` and `GetConnectorProperty`.

2. The connector is asked to initialize.

   This the connector receives a call to the `InitializeConnector` method, which may optionally be overridden. This happens only once during the lifetime of the process.

3. Data source-specific properties are provided to the connector. These have been read from the data source configuration file by the connectivity framework.

   To query these properties, the connector should use the methods `HasConnectorProperty` and `GetConnectorProperty`.

4. TRIPcof searches the TRIP database that is specified in the data source configuration file for items previously imported by the connector.

5. The connector is asked to check if a previously imported item has been changed or deleted in the data source.

   The connector receives a call to the `CheckItem` *method. The* `ImportItem` *argument will contain a UTC timestamp that must be compared to the current value by the connector. The method must return 1 if update is required, 2 if the item can no longer be found in the data source and 0 if the item remains unchanged.*

6. If step 5 indicated that the item has been deleted, TRIPcof will delete the corresponding TRIP record.

7. If step 5 indicated that the item has been modified, the connector will be asked to load the item's actual data and provide an updated timestamp. The TRIP record for the item is updated with the new data.

   The connector receives a call to the `LoadItem` *method. The* `ImportItem` *argument should be populated with the new data for the item and an updated last-modified timestamp.*

8. While there are more records to check, repeat from step 5.

9. The connector is asked to disconnect from the data source.

   The connector receives a call to the `Disconnect` *method. The argument is the object returned from the connector on the call to the* `Connect` *method.*

10. The connector is unloaded when the process shuts down.

    Just prior to this, the connector may receive a call to the `UninitializeConnector` *method, which may optionally be overridden. This happens only once during the lifetime of the process, and after this has occurred, the adapter will no longer be used in the current process.*

Error handling

Whenever an error occurs in your processing such that you cannot continue, you must call `SetError` and throw a `ConnectorException`:

```
SetError(ConnectorReturnCode.InvalidArgument, "Error message here...");
throw new ConnectorException(ConnectorReturnCode.InvalidArgument,
       "Error message here...");
```

An alternative to this is use the `ThrowError` method, which will properly register the error state and throw an exception:

```
ThrowError(ConnectorReturnCode.InvalidArgument, "Error message here...");
```

Failure to call the `SetError` method directly or indirectly as described above may result in TRIPcof getting incorrect information about the status of the current operation. Logging and application feedback may also get confusing and contradictory.

Logging

The connectivity framework supports logging. Connectors can use the `LogDebug` method declared on the `ImportConnector` base class to output debug logging statements. Also, whenever the `SetError` method gets called to register a processing failure, the error message will also be logged, provided that the log level is 2 (errors) or higher.

All logging statements, even ones used for debug purposes, will only be output if the log level of the logging system is higher or equal to the log level of the statement. This means that you can safely leave them in place even in production. The performance hit for checking the logging level for each statement is negligible.

Connector configuration file

All connectors need their own configuration file. This also applies to custom connectors. Please refer to the conf/samples.cfw directory for a template configuration file for a .NET connector (dotnetconnector.conf). It has detailed comments for each setting, so please study it carefully.

Deployed connector configuration files belong in the `conf\cfw.connectors.d` directory under the TRIPcof installation.

A connector written in C# may for instance have the following configuration:

```
[myconnector]
Type=ImportConnector
Enable=True
LabelSample custom connector for .NET
Manufacturer=infinIT Services GmbH
SupportsActiveMonitoring=False
SupportsOnDemandScanning=True
ConnectorLib=..\..\bin\InfinITServices.Trip.ConnectorFramework.ApiWrapper.dll
DotNetAssembly==..\..\bin\Exampe.Trip.Connector.dll
DotNetClass=Exampe.Trip.Connector.MyCustomConnector
WorkDir=..\..\tmp
IsDotNet=True
IsJava=False
LogLevel=5
LogDir=..\..\log
LogPrefix=myconnector
```

Data source configuration file

The configuration of what data the connector should work with and how to reach it must be specified in a separate configuration file.

There are a few properties that all data source configuration files must have:

- Type
- Connector
- Enable
- LocalStore
- StoreCopy

Details on these can be found in section “Setting up a data source for a connector” in this document. Aside for these, the data source configuration should contain any properties needed by the connector to work with a particular data source.

Deployed connector configuration files belong in the `conf\cfw.importds.d` directory under the TRIPcof installation.

Example of a data source configuration for a custom data source, utilizing two custom properties:

```
[myconnector-ds1]
Type=DataSource
Connector=myconnector
Enable=True
LocalStore=CONNECTORDB
MyPropertyOne=Alpha;Beta
MyPropertyTwo=True
```

Deployment

Deploying your custom connector involves:

- Copying the connector assembly DLL to the target machine. You can put this into the bin directory of the TRIPcof installation, or in any directory of your choice.
- Copy the data configuration file for your connector to the directory `conf\cfw.connectors.d` under the TRIPcof installation.
- Copy the data source configuration file(s) for your connector to the directory `conf\cfw.importds.d` under the TRIPcof installation.
- Verify that the data sources for your connector are seen by TRIPcof by running "`cfwimport -list`".

Appendix A: Supported File Formats for Text Extraction

Via Apache Tika (adapter "tikaserver")

- Apple iWorks document formats (Numbers, Pages, Keynote).
- Electronic Publication Format (EPUB)
- HTML and Compressed HTML Help (CHM)
- Microsoft Excel
  versions '97-2007
- Microsoft Word
  versions '97-2007, also limited support for the Word 6 and Word 95 file formats
- Microsoft PowerPoint
  versions '97-2007
- OOXML - Office Open XML (ECMA-376)
  XML-based file format, supported by Microsoft Office 2007 and later
  (file extensions “.docx”, “.pptx” and “.xlsx”)
- OpenDocument format, the default format of OpenOffice and LibreOffice
- WordPerfect WP6+
- QuattroPro QPW v9+
- Portable Document Format (PDF)
- Rich Text Format (RTF)
- XML

Via Microsoft IFilter (adapter "ifilter")

- The formats supported depends on what IFilters are installed on the local Windows machine.
- Please refer to http://en.wikipedia.org/wiki/IFilter for more information and links to other resources.

Appendix B: Supported File Formats for HTML Conversion

Via ICEpdf (adapter "icepdf")

- Portable Document Format (PDF)

Via LibreOffice (adapter “libreoffice”)

- Microsoft Office
    - Versions '97-2007, also limited support for the Word 6 and Word 95
    - OOXML - Office Open XML (ECMA-376), with versions 2007 and later
- OpenDocument format, the default format of OpenOffice and LibreOffice