TrackCore | OPERATING ROOM

TrackCore, Inc.
25 Commerce Avenue SW
Suite 200
Grand Rapids, MI 49503

W implanttracking.com

P 616.632.2222

F 616.632.2225

Integrating With TrackCore

CONFIDENTIAL: This document is indented only for the original recipient. The information contained herein is confidential and may constitutre inside or non-public infomration under international, federal, or state securities laws. Unauthorized forwarding, printing, copying, distribution or use of such information is strictly prohibited and may be unlawful. Please contact TrackCore, Inc. at +1.616.632.2222

TABLE OF CONTENTS

1 EXECUTIVE OVERVIEW 5

1.1 Application Overview 5

1.2 Document Details 5

1.3 Key Factors 5

1.4 Communication Options 5

1.4.1 Barcodes 5
1.4.2 Web Services 6
1.4.3 Remote Interface Client 6

1.5 Data Options 7

1.5.1 Inbound 7
1.5.1.1 Patient Information 7
1.5.1.1.1 TrackCore 4.0 7
1.5.1.1.2 Message Types 8
1.5.1.1.2.1 ADT 8
1.5.1.1.2.2 SIU 9
1.5.1.2 Surgery Case / Implant 9
1.5.1.2.1 Epic Implant Extract File 9
1.5.1.2.2 McKesson Chart Data Message 9
1.5.2 Outbound 9
1.5.2.1 Overview 9

2 USING THE TRACKCORE WEB SERVICES 10

2.1 Process Overview 10

2.2 URLS 11

2.3 Web Methods 11

2.3.1 /RIH/RemoteInterfaceHost.svc/RegisterInterfaceClient 11
2.3.1.1 Purpose 11
2.3.1.2 Input Parameters 11
2.3.1.3 Output 12
2.3.1.4 Sample Call 12
2.3.2 /RIH/RemoteInterfaceHost.svc/GetEncryptionInfo 13
2.3.2.1 Purpose 13
2.3.2.2 Input Parameters 13
2.3.2.3 Output 13
2.3.2.4 Sample Call 13
2.3.3 /RIH/RemoteInterfaceHost.svc/RevokeInterfaceClient 14
2.3.3.1 Purpose 14
2.3.3.2 Input Parameters 14
2.3.3.3 Output 14
2.3.3.4 Sample Call 15

2.3.4 /RIH/RemoteInterfaceHost.svc/SendData 15
2.3.4.1 Purpose 15
2.3.4.2 Input Parameters 15
2.3.4.3 Output 16
2.3.4.4 Sample Call 16
2.3.5 /RIH/RemoteInterfaceHost.svc/GetData 18
2.3.5.1 Purpose 18
2.3.5.2 Input Parameters 18
2.3.5.3 Output 18
2.3.5.4 Sample Call 18

3 USING THE REMOTE INTERFACE CLIENT (RIC) 20

3.1 Process Overview 20

3.2 Installing And Configuring the Client 21
3.2.1 System Requirements 21
3.2.2 Installation Directions 21
3.2.3 Registering the Client 21

3.3 Using RIC to Transfer Files 24

3.4 Communicating PORT to PORT using TCP/IP 24

1 EXECUTIVE OVERVIEW

1.1 APPLICATION OVERVIEW

TrackCore is a web-based application that assists health organizations in their efforts to meet The Joint Commission and FDA standards for implant tracking. This document provides healthcare organizations with the tools necessary to integrate existing hospital information systems with TrackCore in order to achieve streamlined processes in their efforts to meet compliance.

1.2 DOCUMENT DETAILS

Beyond the executive overview, this document is intended primarily for IT departments that wish to integrate existing data with TrackCore using HL7, web service, file transfer and barcodes. Much of this document is technical in nature and geared towards technical systems integration specialists. Other users may want to reference the User Manual or discuss integration options with an implementation specialist or TrackCore account representative to find out how this and other integration options can result in cost and data entry error savings.

1.3 KEY FACTORS

There are several key factors when considering an interface.

- The data output, as the result of an interface, is only as good as the data passed in to the interface. Every effort should be made to achieve data standardization throughout the enterprise.
- Many fields in the integrations are optional; however, by not transmitting them to the system, they will not be available to the end users, limiting the benefits of the integration.
- TrackCore interfaces are designed to streamline the data entry process, and as such, treat data input from an interface as if the source system pre-validated the data. This means, with few exceptions, there is little data validation that occurs in the interface.
- The primary delivery model for TrackCore is that of Software as a Service (SaaS), where we (TrackCore) host the servers at our secure hosting facility and the hospital uses a standard web browser to access the primary functionality of the application.
- All integrations are completed via the internet. All transactions, including integrations, are secured using the HTTPS protocol (128 bit encryption using an SSL certificate). This delivery method is one of the factors that drive the technology used by our integration platform.
- Facilities that do not have IT implementation teams may also consult with an implementation specialist for ways they can integrate their existing systems with TrackCore using this integration platform.
- This document will be periodically updated as additional integration options are made available.
- If you are interested in integrating in ways not mentioned here, contact softwaresupport@tissuetrackcore.com.

1.4 COMMUNICATION OPTIONS

1.4.1 Barcodes

TrackCore allows hospitals to streamline data entry using standard barcode technologies in several ways. For the most part, barcode use is handled through direct conversation with our customer service team, however, a brief overview is included here as some barcode use impacts certain machine-to-machine interfaces by allowing the data encoded in the barcode to close the loop between systems where two-way machine-to-machine communication is not possible or feasible. In these scenarios, both 1D and 2D barcodes in a wide variety of symbologies are widely supported.

1. Vendor and product information encoded in product barcodes using GS1, HIBC and ISBT 128 barcode formats is parsed by TrackCore upon receipt of an item and the information is used to populate important regulatory fields like expiration date, part number, serial number, etc.

    a. Where a part number or other device identifier is determined that information is used to lookup items in the product master to retrieve their data. (*This capability is available in TrackCore 4.0 and later and is automatically included for all customers.*)

    b. Where a serial number is determined, certain vendor level integrations also allow cases where production level information like lot number, cost, shipment, etc. to be imported directly into TrackCore for that specific item. (*This capability is available in TrackCore 3.5 and later and is automatically included for all customers for certain implant vendors. Contact TrackCore support for a list of vendors that provide data for this integration.*)

2. At the point where an item is associated in TrackCore with a patient, patient barcodes included on charts or wrist bands, can be scanned in order to retrieve an MRN, encounter number, case number, account number or the like and that information can be used to lookup the patient information necessary to complete the record. (*This capability is available in TrackCore 3.5 and later. Contact TrackCore support for information on how to set this up.*)

3. Upon receipt of an item into the TrackCore system, we strongly advise our customers to label the items with TrackCore-specific labels. These labels both guarantee uniqueness of the item tracked and at the same time provide a visual indicator to everyone that the item is tracked in TrackCore. These labels can be programmed to include specific product information that simplifies scanning product information into other health information systems (HIS). (*This capability is available in TrackCore 3.5 and later. Contact TrackCore support for information on how to set this up.*)

1.4.2 Web Services

Secure web services serve as the foundation for all TrackCore interfaces. A **Web service** is a method of communications between two electronic devices over the World Wide Web. It is a software function provided at a network address over the web with the service always on.[1] The web services allow a facility to send or receive HL7 messages or files to/from TrackCore in a secure manner and allows for the processing of the message to occur within TrackCore. Hospitals with the capabilities to access and consume web services directly are able to do so. The directions and methods for doing this are provided in Using The TrackCore Web Services section of this document. Directions on mapping data for specific data interfaces are included in the section labeled Data Options.

For those facilities without the tools or bandwidth to consume the web services are able to use the TrackCore Remote Interface Client (RIC) described in the next section. Since the RIC requires a virtual machine or PC to host it, the direct use of web services offers the hospital the ability to save on long-term hardware and IT costs. At the same time, the RIC offers hospitals the ability to reduce the upfront effort and costs of integration.

1.4.3 Remote Interface Client

The TrackCore Remote Interface Client (RIC) is a lightweight Windows service and application that reduces a hospital's upfront interface development work by serving as a secure transport mechanism for files as well as HL7 messages and files. The RIC receives information from the internal hospital information systems, encrypts that data and then securely

[1] Wikipedia - http://en.wikipedia.org/wiki/Web_service

transmits it to the TrackCore Web Services, effectively doing most of the technical heavy lifting for the hospital interface team.

While there is a charge for interfaces, the charge is for the web services aspect of the interface and the RIC is provided to hospitals at no charge. As such, a hospital can switch between using the RIC or directly consuming the web services at any time.

The Remote Interface Client (RIC) can be configured to listen to a specific TCP/IP port on the host machine or monitor a specific directory for messages. Your IT staff will need to determine how your organization will provide the messages.

It is the responsibility of your organization to send the HL7 messages to the correct port or to put the messages in the appropriate directory. Likewise, for inbound messages the hospital is responsible for listening for messages on the correct IP and port or for monitoring the configured directory for inbound files.

Detail on implementing the client is provided in the section labeled Using the Remote Interface Client (RIC). Directions for mapping data on specific interfaces are included in the next section.

1.5 DATA OPTIONS

The TrackCore interface system facilitates dynamic mapping between varieties of message formats. The following list breaks down data transfer options by direction; Outbound (from TrackCore) and Inbound (to TrackCore); and by the type of data transferred. Once the desired direction and type of data is determined, reference the associated mapping template file and specification (where provided). Each mapping file provides field definitions, field requirements and recommended default source fields.

Dynamic mapping allows every hospital to use existing message formats and send those to TrackCore with minimal (if any) changes to the message before sending to TrackCore. For each interface that a hospital uses, their interface team should provide a copy of the mapping file with their changes. The TrackCore interface implementation team will use this file to define the interface prior to implementation.

If your facility desires to transmit or receive data that is not mentioned below, please contact a TrackCore support representative or account manager to discuss custom interface options.

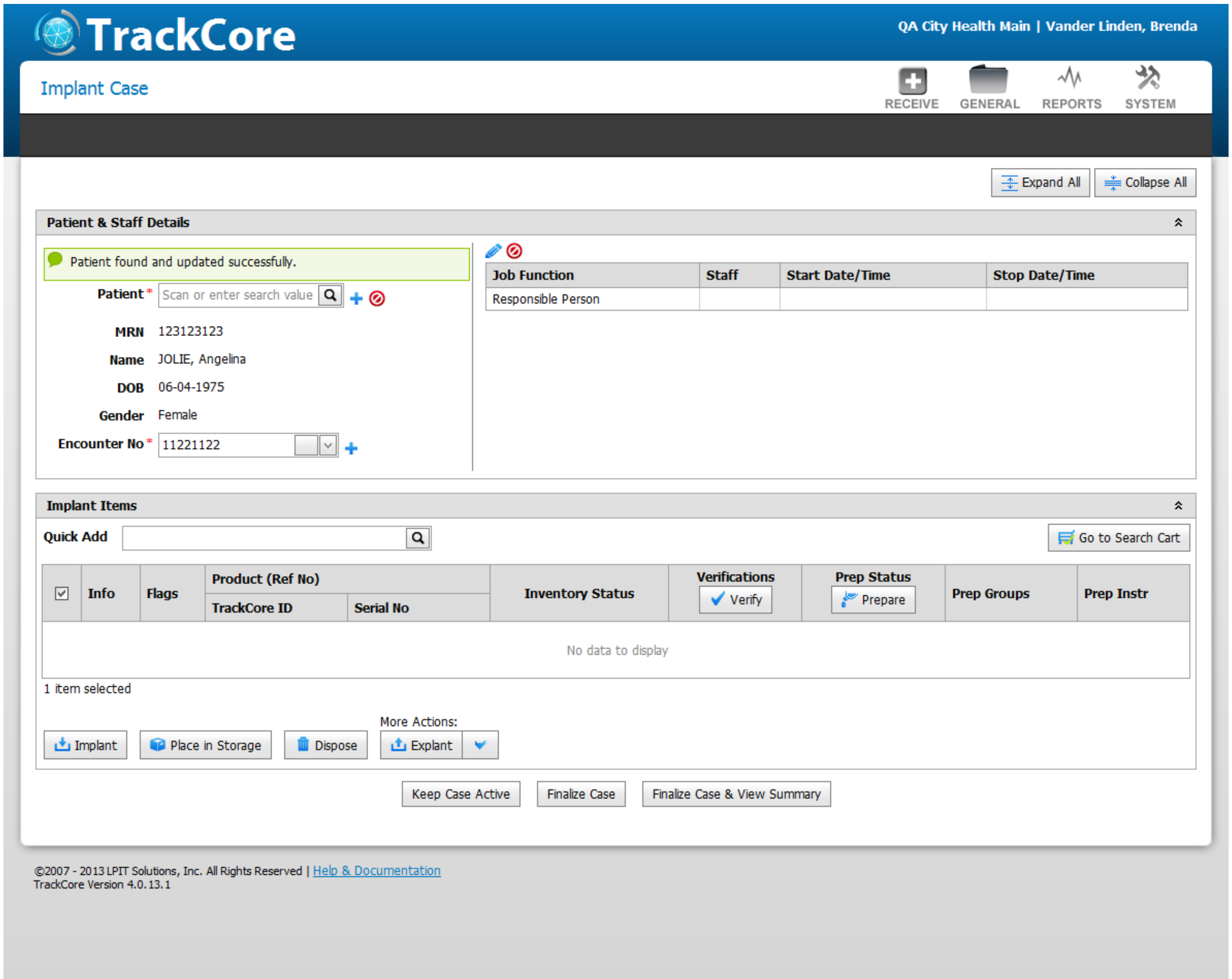
1.5.1 Inbound

1.5.1.1 PATIENT INFORMATION

The inbound patient information interface allows a hospital to send patient information to TrackCore which is used when entering an implant record in TrackCore. The end user scans, or keys in, a patient identifier on the case / implant screen which is then used to retrieve the patient information from a holding table and associate it with the implants on the TrackCore screen.

1.5.1.1.1 TrackCore 4.0

In TrackCore 4.0 and later, when the Patient Information interface is implemented, the user is able to key in or scan either the patient identifier (MRN) or an encounter number and have the appropriate patient information be pulled into the implant record for verification and then inclusion in the final record. In the screen shot below, scanning of the encounter number resulted in population of the patient MRN, name, DOB and Gender. Depending on the message sent and the configuration of TrackCore, other information may also be captured such as surgeon and staff working on the case, patient address and the type of procedure. Generally speaking, SIU (Scheduling) and ADT (Admit) type HL7 messages are sent, if the hospital has a different message structure for sending patient information, a custom interface can be developed. SIU (Scheduling) messages tend to be the most relevant and contain the most information that is pertinent to an implant case. Reference the mapping documents contained within each section to see what fields are available to be transmitted to TrackCore.



TrackCore

QA City Health Main | Vander Linden, Brenda

Implant Case

Patient & Staff Details

Patient found and updated successfully.

Patient*	Scan or enter search value
MRN	123123123
Name	JOLIE, Angelina
DOB	06-04-1975
Gender	Female
Encounter No*	11221122

Job Function	Staff	Start Date/Time	Stop Date/Time
Responsible Person			

Implant Items

Quick Add

☑	Info	Flags	Product (Ref No)		Inventory Status	Verifications	Prep Status	Prep Groups	Prep Instr
			TrackCore ID	Serial No		✓ Verify	Prepare		
				No data to display					

1 item selected

More Actions:

©2007 - 2013 LPIT Solutions, Inc. All Rights Reserved | Help & Documentation
TrackCore Version 4.0.13.1

1.5.1.1.2 Message Types

1.5.1.1.2.1 ADT

- ADT_A01 – Admit/visit Notification (Inserts new patient scheduling data): TrackCore Integrations - ADT_A01 Mapping.xlsx

- ADT_A04 – Registration (Inserts new patient scheduling data): TrackCore Integrations - ADT A04 Mapping.xlsx

- ADT_A08 – Update Admit/visit (Updates patient scheduling data): TrackCore Integrations - ADT A08 Mapping.xlsx

- ADT_A11 – Cancel Admit/visit (Removes patient scheduling data): TrackCore Integrations - ADT A11 Mapping.xlsx

1.5.1.1.2.2 SIU

- SIU_S12 – Notification of new appointment booking (Inserts new patient scheduling data): TrackCore Integrations - SIU_S12 Mapping.xlsx

- SIU_S13 – Notification of appointment rescheduling (Updates patient scheduling data): TrackCore Integrations - SIU_S13 Mapping.xlsx

- SIU_S14 – Notification of appointment modification (Updates patient scheduling data): TrackCore Integrations - SIU_S14 Mapping.xlsx

- SIU_S15 – Notification of appointment cancellation (Removes patient scheduling data): TrackCore Integrations - SIU_S15 Mapping.xlsx

1.5.1.2 Surgery Case / Implant

The inbound surgery case / implant interface allows a hospital to enter information related to a case / procedure to TrackCore and have that information complete a TrackCore case / implant log in order to complete the chain of custody for a tissue or implant. This interface reduces the need for a hospital to have its staff enter implant information in both the EHR perioperative record as well as the TrackCore implant screen. Most EHR systems do not capture all of the fields necessary to complete the chain of custody in an FDA / The Joint Commission compliant manner. However, in TrackCore version 4.0 and higher, incomplete data can be sent to TrackCore and the system will allow users at a facility to log in to TrackCore after the fact and complete only the data that is missing from a compliant record.

1.5.1.2.1 Epic Implant Extract File

This interface is only available for facilities that are using Epic OpTime 2012 or later for their EHR and TrackCore 3.5 or later for their implant and tissue tracking.

- Epic to TrackCore 3.5 Mapping Document: TrackCore Integrations - Epic to TrackCore 3.5 Mapping.xlsx

- Epic to TrackCore 4.x Mapping Document: TrackCore Integrations - Epic to TrackCore 4.0 Mapping.xlsx

- Overview and Epic and TrackCore configuration: TrackCore Integrations - Epic Implant Interface.docx

1.5.1.2.2 McKesson Chart Data Message

This interface is available with the McKesson Chart Data Message which is an HL7 message specification based on the ORU_R01 definition. This interface will work with facilities using McKesson and TrackCore 4.0 or later for their implant and tissue tracking.

- Overview and McKesson and TrackCore application configuration: TrackCore Integrations - McKesson Implant Interface.docx

- McKesson Chart Data Specifications: 16.0 Chart Data Message Specifications.pdf

- ORU_R01 McKesson to TrackCore 4.x Mapping Document: TrackCore Integrations - ORU_R01 McKesson Mapping.xlsx

1.5.2 Outbound

1.5.2.1 Overview

Messages and files can also be sent from TrackCore to hospital information systems. Currently there are no outbound interfaces developed and are generally considered custom development available upon request. Samples of such interfaces would include Item Consumption, Inventory Restock reports and Outbound Case reports. Please contact a TrackCore account representative for more information.

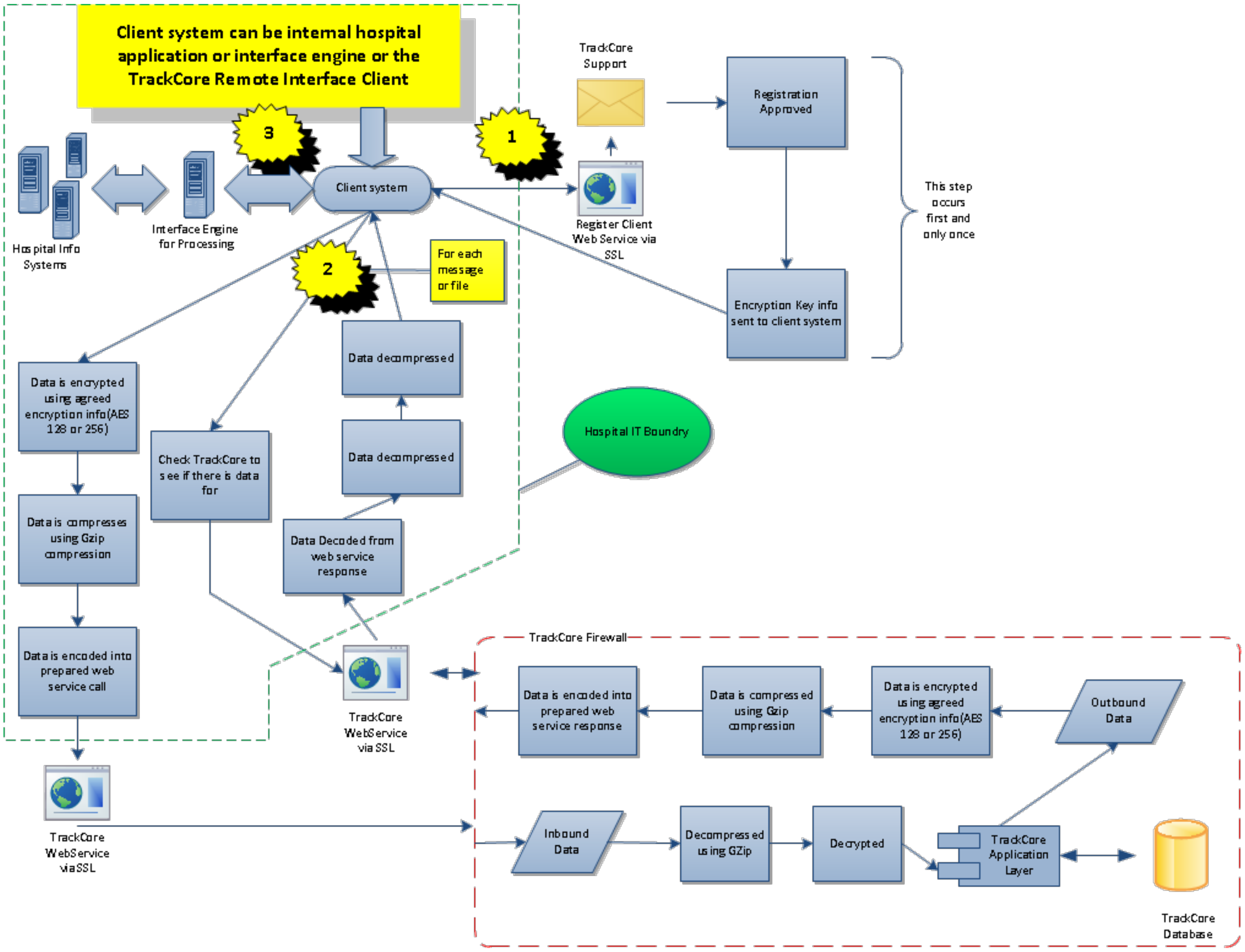
2 Using The TrackCore Web Services

2.1 PROCESS OVERVIEW

The TrackCore Web Services start by establishing a trust relationship between the client system and the web services system. This involves a web service call made by the client system that registers the interface. Once the TrackCore team approves the registration, the client system is sent a revocable AES encryption key and initialization vector. These two pieces of information are used to encrypt and decrypt all of the data for that facility.

When the client system sends data to TrackCore, it encrypts the data and then uses a GZip compression library to compress the data. The data is then encoded into the web services message to TrackCore. TrackCore decompresses the data, unencrypts it and process it into the TrackCore system. When TrackCore sends data to the client system, the reverse process is used. This process is illustrated in the following diagram.

Reference section 1.5 of this document for the different kind of data that can be transmitted and received.


Client system can be internal hospital application or interface engine or the TrackCore Remote Interface Client
3
1
TrackCore Support
Registration Approved
This step occurs first and only once
Hospital Info Systems
Interface Engine for Processing
Client system
Register Client Web Service via SSL
2
For each message or file
Encryption Key info sent to client system
Data decompressed
Data decompressed
Hospital IT Boundry
Data is encrypted using agreed encryption info(AES 128 or 256)
Check TrackCore to see if there is data for
Data is compresses using Gzip compression
Data Decoded from web service response
Data is encoded into prepared web service call
TrackCore WebService via SSL
TrackCore Firewall
Data is encoded into prepared web service response
Data is compressed using Gzip compression
Data is encrypted using agreed encryption info(AES 128 or 256)
Outbound Data
TrackCore WebService via SSL
Inbound Data
Decompressed using GZip
Decrypted
TrackCore Application Layer
TrackCore Database

The client system can be either the TrackCore Remote Interface Client described in section 3 or it can be an internal hospital application or a module that is a part of the hospital's interface engine. Reference the Executive Overview section of this document for the pros and cons of each approach.

2.2 URLS

For the purposes of testing interfaces from the hospital to TrackCore, the domain portion of the URL should be https://trackcore4wsca.lpitsolutions.com

For the purposes of product interfaces from the hospital to TrackCore, the domain portion of the service URL should be https://trackcore4webservice.lpitsolutions.com

From this point in the document forward, the documentation will presume these to be the initial part of the URL and it will not be displayed.

2.3 WEB METHODS

The interfaces are setup as serialized JSON calls and use standard HTTP Post methods. Sample code provide is in .NET C#. The use of JSON and HTTP Post allows a variety of languages and platforms to be used to complete the integration, however the customer is responsible for programming, debugging and supporting their own code if they choose to use other languages or means of consuming the web service.

This section only addresses how to utilize the web methods to send and receive data. The services are designed to send and receive entire messages or files. There is no data mapping necessary as a result of the interface. Reference section 1.5 to identify what data can be sent and what, if any, data mapping is necessary.

2.3.1 /RIH/RemoteInterfaceHost.svc/RegisterInterfaceClient

2.3.1.1 PURPOSE

RegisterInterfaceClient creates an account for the hospital that allows them to communicate to TrackCore through our RemoteInterfaceHost webservice.

2.3.1.2 INPUT PARAMETERS

- UserName: The first authentication element for the sending application.
- Password: The second authentication element for the sending application.
- IPAddress: This is the public IP address of the application calling the interface. This serves as a third form of authentication as traffic from outside this IP Address to the web service is ignored.
- Company: To verify the hospital identity and for the purposes of verifying the right to use the interface and to determine who should be contacted should there be an issue with the interface.
- Email: Email address of the primary contact for the interface at the customer site.
- Phone: Phone number of the primary contact for the interface at the customer site.
- ContactName: Name of the primary contact for the interface at the customer site.
- Address: Address of the company for verification purposes.
- EffectiveStartDate: Date that use (including testing) of the web service is slated to begin.
- EffectiveEndDate: Date that use (including testing) of the web services is scheduled to conclude. This date can be set for up to 10 years from the EffectiveStartDate.

- DefaultInboundFileDestinationPath: (Future)
- CallBackServicePath: (Future)

2.3.1.3 Output

Serialized as JSON

```
        private class GenericResponse
        {

            public Boolean Success { get; set; }


            public string Message { get; set; }
        }
```

2.3.1.4 Sample Call

```
var req = HttpWebRequest.Create(string.Format("{0}/RegisterInterfaceClient", _uriMap[(string)server.Select
edItem]));
                        req.ContentType = "application/json";
                        req.Method = "POST";

                        new DataContractJsonSerializer(typeof(RegisterInterfaceClientRequest)).WriteObject
(req.GetRequestStream(), new RegisterInterfaceClientRequest
                        {
                            Username = userName.Text.Trim(),
                            Password = password.Text.Trim(),
                            IPAddress = ipAddress.IPAddress,
                            Company = company.Text.Trim(),
                            Email = email.Text.Trim(),
                            Phone = phone.Text.Trim(),
                            ContactName = contactName.Text.Trim(),
                            Address = address.Text.Trim(),
                            EffectiveEndDate = DateTime.Parse(endDate.Text),
                            EffectiveStartDate = DateTime.Parse(startDate.Text),
                            DefaultInboundFileDestinationPath = string.Empty,
                            CallBackServicePath = string.Empty
                        });

                        var resp = (HttpWebResponse)req.GetResponse();
                        switch (resp.StatusCode)
                        {
                            case HttpStatusCode.OK:
                                var result = new DataContractJsonSerializer(typeof(RegisterInterfaceClient
Response)).ReadObject(resp.GetResponseStream()) as RegisterInterfaceClientResponse;
                                if (result != null && result.Success)
                                {
                                }
                                break;

                            default:
                                break;
                        }


        private class RegisterInterfaceClientRequest
        {

            public string Username { get; set; }

            public string Password { get; set; }

            public string Company { get; set; }
            public string ContactName { get; set; }

            public string Address { get; set; }

            public string Email { get; set; }

            public string Phone { get; set; }

            public string IPAddress { get; set; }

            public DateTime EffectiveStartDate { get; set; }

            public DateTime EffectiveEndDate { get; set; }

            public string DefaultInboundFileDestinationPath { get; set; }

            public string CallBackServicePath { get; set; }
        }
```

2.3.2 /RIH/RemoteInterfaceHost.svc/GetEncryptionInfo

2.3.2.1 Purpose

This method returns the 256 bit AES Encryption Key and Initialization Vector to be used when encrypting and decrypting data sent using the interface. This method will only return the Key and Vector as long as the encryption key is not locked. They key will become locked when SendData or GetData is successfully used to encrypt/decrypt information. This information is available after the Remote Interface Client is reviewed and approved by the TrackCore implementation team. Programs that consume the web service are responsible for storing and retrieving the Key and Vector and using them to encrypt the information sent.

2.3.2.2 Input Parameters

- Username: Username provided when the Client was registered.
- Password: Password provided when the Client was registered.
- (Sending IP Address is automatically captured for additional verification.)

2.3.2.3 Output

```
    public class EncryptionInfo
    {

        public byte[] Key { get; set; }


        public byte[] IV { get; set; }
    }
```

2.3.2.4 Sample Call

```
                var req = HttpWebRequest.Create(string.Format("{0}/GetEncryptionInfo", _config.Uri));
                req.ContentType = "application/json";
                req.Method = "POST";

                // Serialize the data into the request
                new
DataContractJsonSerializer(typeof(GetEncryptionInfoRequest)).WriteObject(req.GetRequestStream(), new
GetEncryptionInfoRequest
                {
                    Username = _config.UserName,
                    Password = _config.Password
                });

                // Execute the request
                var resp = (HttpWebResponse)req.GetResponse();
                switch (resp.StatusCode)
                {
                    case HttpStatusCode.OK:
                        var result = new
DataContractJsonSerializer(typeof(GetEncryptionInfoResponse)).ReadObject(resp.GetResponseStream()) as
GetEncryptionInfoResponse;
                        if (result.Key != null && result.IV != null)
                        {
                            byte[] key;

                            using (var ms = new MemoryStream())
                            {
                                new BinaryFormatter().Serialize(ms, new EncryptionInfo(result.Key,
result.IV));

                                key = ProtectedData.Protect(ms.ToArray(), _keyEntropy.ToByteArray(),
DataProtectionScope.LocalMachine);
                            }

                            using (var fs = new FileStream(keyFile, FileMode.Create, FileAccess.Write))
                            {
                                fs.Write(key, 0, key.Length);
                            }

                            _logger.Log("REQUEST KEY", "Key retrieved", true);
                        }
                        else
                        {
                            _logger.Log("REQUEST KEY", "Key not available", false);
                        }
                        break;

                    default:
                        // TODO:  Figure out what to do if it fails
                        _logger.Log("REQUEST KEY", "Request failed", false);
                        break;
                }
```

2.3.3 /RIH/RemoteInterfaceHost.svc/RevokeInterfaceClient

2.3.3.1 Purpose

The purpose of this call is to discontinue use of this Remote Interface Client instance. This might be done for security purposes, discontinuation of service, etc.

2.3.3.2 Input Parameters

- Username: Username provided when the Client was registered.
- Password: Password provided when the Client was registered.
- EffectiveEndDate: The date when you want the discontinuation to be effective.
- (Sending IP Address is automatically captured for additional verification.)

2.3.3.3 Output

Serialized as JSON

```
private class GenericResponse
{

    public Boolean Success { get; set; }
```

```
            public string Message { get; set; }
        }
```

2.3.3.4 Sample Call

```
var req = HttpWebRequest.Create(string.Format("{0}/RevokeInterfaceClient", _uriMap[(string)server.Selected
Item]));
                        req.ContentType = "application/json";
                        req.Method = "POST";

                        new DataContractJsonSerializer(typeof(RegisterInterfaceClientRequest)).WriteObject
(req.GetRequestStream(), new RevokeInterfaceClientRequest
                        {
                            Username = userName.Text.Trim(),
                            Password = password.Text.Trim(),
                            EffectiveEndDate = DateTime.Parse(endDate.Text)
                        });

                        var resp = (HttpWebResponse)req.GetResponse();
                        switch (resp.StatusCode)
                        {
                            case HttpStatusCode.OK:
                                var result = new DataContractJsonSerializer(typeof(RevokeInterfaceClientRe
sponse)).ReadObject(resp.GetResponseStream()) as RevokeInterfaceClientResponse;
                                if (result != null && result.Success)
                                {
                                }
                                break;

                            default:
                                break;
                        }
```

2.3.4 /RIH/RemoteInterfaceHost.svc/SendData

2.3.4.1 Purpose

SendData is used to send an entire message in the Data field. This message can be an HL7 message, flat file, etc. This method is available after the Remote Interface Client is reviewed and approved by the TrackCore implementation team.

2.3.4.2 Input Parameters

- Username: Username provided when the Client was registered.

- Password: Password provided when the Client was registered.

- FacilityID: The FacilityID is one of two indicators of the hospital sending the information. This allows one interface client (identified with the UserName and Password) to send messages from multiple facilities within the same system and have the information sent to the correct TrackCore site. The TrackCore team provides this.

- IntegrationCode: This is the second of the two indicators of the hospital sending the information. The TrackCore implementation team also provides this.

- MessageType: This is a 3-character indicator of the type of data that is being sent. For HL7 messages, this relates directly to the MSH.9.1 field. For flat files or non-HL7, the TrackCore interface implementation team provides this value.

- EventType: This is a 3-character indicator of the action that the data represents (ie Create, Read, Update or Delete). For HL7 messages, this relates directly to the MSH.9.2 field. For flat files or non-HL7, the TrackCore interface implementation team provides this value.

- Data: Byte array of the data being sent using the interface. This data must be GZipped and AES encrypted before being sent.

- (Sending IP Address is automatically captured for additional verification.)

2.3.4.3 Output

```
        private class GenericResponse
        {

            public Boolean Success { get; set; }


            public string Message { get; set; }
        }
```

2.3.4.4 Sample Call

This example processes one or more specifically named files within a directory, reads the data in and then sends it. The data in this example is encrypted within the file.

//The following block, reads a file into a stream, uses GZIP to compress the data and then encrypts it using the Key and Vector. The same can be done with a string prior to transmittal.

```
                        using (var fs = new FileStream(fileToProcess, FileMode.Open, FileAccess.Read, 
FileShare.None))
                        {
                            // Generate the name for the file
                            var folder = 
_config.MonitoredFolders.Cast<MonitoredFolderConfigurationElement>().Single(f => 
f.Path.Equals(Path.GetDirectoryName(fileToProcess)));

                            using (var fsOut = new FileStream(Path.Combine(_config.OutboundProcessingPath, 
string.Join("_", new[] { folder.FacilityID, folder.IntegrationCode, folder.MessageType, folder.EventType, 
Guid.NewGuid().ToString().Replace("-", "") })).ToUpperInvariant(), FileMode.Create, FileAccess.Write, 
FileShare.None))
                            using (var gzip = new GZipStream(fsOut, CompressionMode.Compress))
                            using (var aes = new AesManaged())
                            using (var enc = aes.CreateEncryptor(_encryptionInfo.Key, 
_encryptionInfo.Vector))
                            using (var cs = new CryptoStream(gzip, enc, CryptoStreamMode.Write))
                            {
                                int bytesRead = 0;
                                byte[] data = new byte[65536];

                                while ((bytesRead = fs.Read(data, 0, data.Length)) != 0)
                                {
                                    cs.Write(data, 0, bytesRead);
                                }
                            }
                        }
```

//file is now transmitted in a separate procedure.

```
var files = Directory.GetFiles(_config.OutboundProcessingPath).Select(f => new FileInfo(f)).OrderBy(f => 
f.CreationTime).ToList();
                    files.ForEach(f =>
                    {
                        try
                        {
                            var msgParams = f.Name.Split('_');

                            // Define the data to post
                            var postData = new SendDataRequest
                            {
                                Username = _config.UserName,
                                Password = _config.Password,
                                FacilityID = msgParams[0],
                                IntegrationCode = msgParams[1],
                                MessageType = msgParams[2],
                                EventType = msgParams[3],
                            };

                            using (var fs = f.OpenRead())
                            {
                                int bytesRead;
                                var data = new byte[65535];

                                while ((bytesRead = fs.Read(data, 0, data.Length)) != 0)
                                {
                                    postData.Data.AddRange(data.Take(bytesRead));
                                }
                            }

                            var req = HttpWebRequest.Create(string.Format("{0}/SendData", _config.Uri));
                            req.ContentType = "application/json";
                            req.Method = "POST";

                            // Serialize the data into the request
                            new
DataContractJsonSerializer(typeof(SendDataRequest)).WriteObject(req.GetRequestStream(), postData);

                            // Submit the request
                            var resp = (HttpWebResponse)req.GetResponse();
                            switch (resp.StatusCode)
                            {
                                case HttpStatusCode.OK:
                                    var result = new
DataContractJsonSerializer(typeof(SendDataResponse)).ReadObject(resp.GetResponseStream()) as
SendDataResponse;
                                    if (result != null && result.Success)
                                    {
                                        // Delete the file
                                        f.Delete();

                                        _logger.Log(Outbound.ACTION, f.FullName, true);
                                    }
                                    else
                                    {
                                        _logger.Log(Outbound.ACTION, string.Format("{0}{1}{2}",
f.FullName, Environment.NewLine, result != null ? result.Message : "No Result"), false);
                                    }
                                    break;

                                default:
                                    _logger.Log(Outbound.ACTION, f.FullName, false);
                                    break;
                            }
                        }
                        catch
                        {
                            _logger.Log(Outbound.ACTION, f.FullName, false);
                        }
                    });
```

2.3.5 /RIH/RemoteInterfaceHost.svc/GetData

2.3.5.1 Purpose

The GetData method determines if there is any data available for the requesting client and if so provides it in the DataContent in an encrypted and zipped manner.

2.3.5.2 Input Parameters

- Username: Username provided when the Client was registered.
- Password: Password provided when the Client was registered.
- (Sending IP Address is automatically captured for additional verification.)

2.3.5.3 Output

```
    public class DataResponse
    {

        public string ResponseType { get; set; }  //EncKeyAvailable, SoftwareUpdate, NoData, Data


        public string ResponseMessage { get; set; } //If ResponseType == SoftwareUpdate, this contains the
path to download update.

         public DataContent[] DataContent { get; set; } //Can contain one or more messages for this
response type.
    }
```

2.3.5.4 Sample Call

//The following call gets the data from the server and writes it out to a local file. You will want to unencrypt the byte array and then use GZip to unzip it.

```
                       var req = HttpWebRequest.Create(string.Format("{0}/GetData", _config.Uri));
                        req.ContentType = "application/json";
                        req.Method = "POST";

                        // Serialize the data into the request
                        new
DataContractJsonSerializer(typeof(GetDataRequest)).WriteObject(req.GetRequestStream(), new GetDataRequest
                        {
                            Username = _config.UserName,
                            Password = _config.Password
                        });

                        // Execute the request
                        var resp = (HttpWebResponse)req.GetResponse();
                        switch (resp.StatusCode)
                        {
                            case HttpStatusCode.OK:
                                var result = new
DataContractJsonSerializer(typeof(GetDataResponse)).ReadObject(resp.GetResponseStream()) as
GetDataResponse;
                                if (result != null)
                                {
                                    switch (result.ResponseType.ToUpperInvariant())
                                    {
                                        case "DATA":
                                            result.DataContent.ForEach(d =>
                                            {
                                                var fileName = Path.Combine(_config.InboundProcessingPath,
d.DestinationFileName);
                                                try
                                                {
                                                    _logger.Log(Inbound.ACTION, fileName, true);
                                                }
                                                catch
                                                {
                                                    _logger.Log(Inbound.ACTION, fileName, false);
                                                }
                                            });
                                            break;

                                        case "ENCKEYAVAILABLE":  // Retrieval of key is started prior to 
inbound processor being started.
                                            break;

                                        case "NODATA":
                                            break;

                                        case "SOFTWAREUPDATE":
                                            break;

                                        case "ERROR":
                                            _logger.Log(Inbound.ACTION, result.ResponseMessage, false);
                                            break;

                                        default:
                                            break;
                                    }
                                }
                                else
                                {
                                    // handle an unsuccessful request
                                }
                                break;

                            default:
                                // handle a request failure
                                break;
                        }
```

3 USING THE REMOTE INTERFACE CLIENT (RIC)

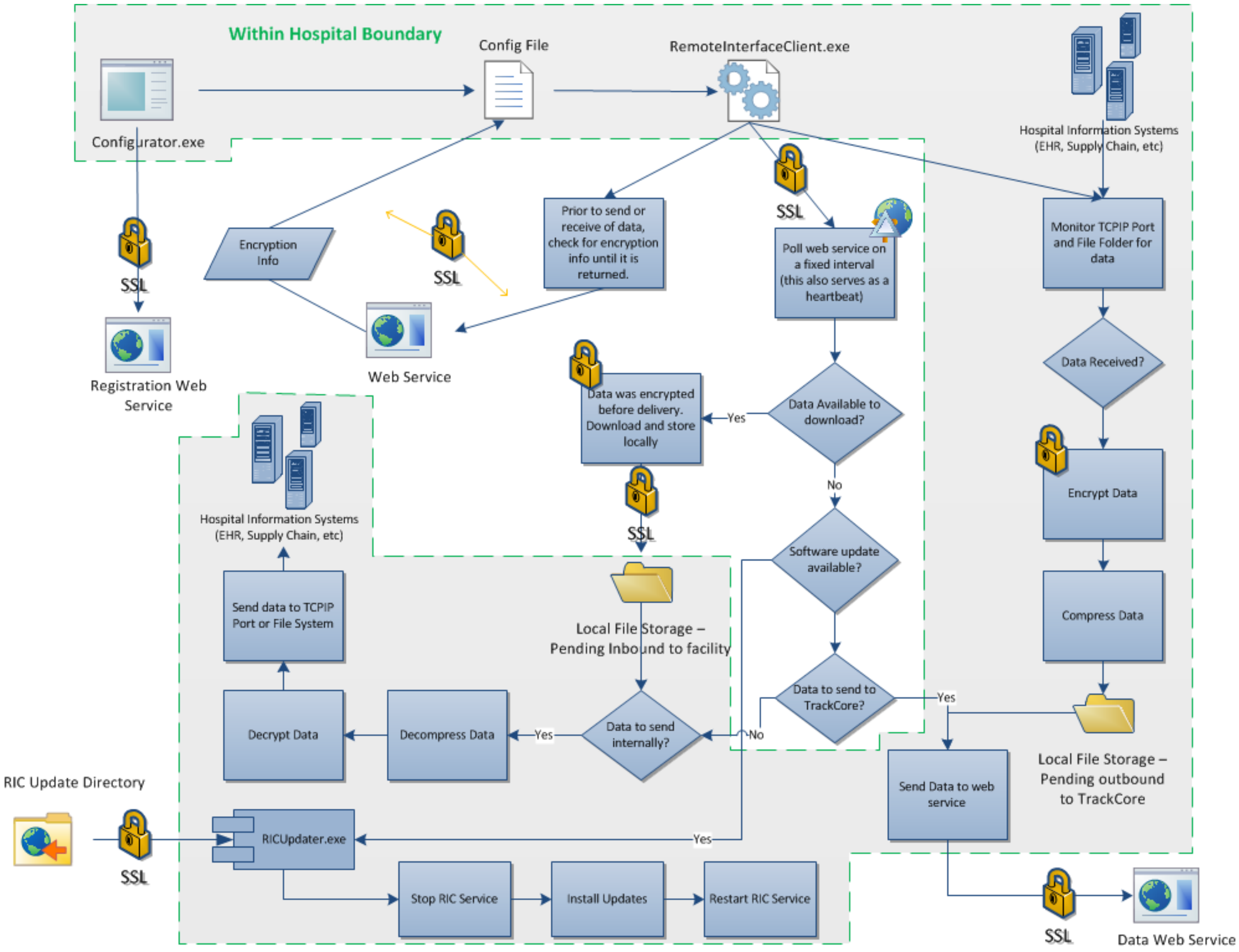
3.1 PROCESS OVERVIEW

The RIC is comprised primarily of three separate programs that work in concert to provide the capability of transferring data between a hospitals internal computing systems and TrackCore servers. These components are:

Configurator.exe: This is a Windows form that allows both registration of the client with the TrackCore Remote Interface Host web service as well as local configuration of the service.

RemoteInterfaceClient.exe: This is a Windows service application that is runs automatically and is capable of monitoring both TCP/IP ports and folders for files. This piece of the application is responsible for receiving data, encrypting / decrypting it and transmitting it either to TrackCore servers or to internal servers for processing.

RICUpdater.exe: This is a transparent console application which is executed by RemoteInterfaceClient.exe when it receives a message that an application update is available. This application is responsible for downloading the application update, shutting down the windows service, installing the update and then restarting the windows service.



Within Hospital Boundary
Config File
RemoteInterfaceClient.exe
Configurator.exe
Hospital Information Systems (EHR, Supply Chain, etc)
SSL
Encryption Info
SSL
Prior to send or receive of data, check for encryption info until it is returned.
SSL
Poll web service on a fixed interval (this also serves as a heartbeat)
Monitor TCPIP Port and File Folder for data
Registration Web Service
Web Service
Data Received?
Data was encrypted before delivery. Download and store locally
Yes
Data Available to download?
Hospital Information Systems (EHR, Supply Chain, etc)
SSL
No
Encrypt Data
Software update available?
Send data to TCPIP Port or File System
Local File Storage – Pending Inbound to facility
Compress Data
Data to send to TrackCore?
Yes
Decrypt Data
Decompress Data
Yes
Data to send internally?
No
Local File Storage – Pending outbound to TrackCore
RIC Update Directory
Send Data to web service
SSL
RICUpdater.exe
Yes
Stop RIC Service
Install Updates
Restart RIC Service
SSL
Data Web Service

3.2 INSTALLING AND CONFIGURING THE CLIENT

3.2.1 System Requirements

It is possible to run this application along with other applications on the same host, however, it is highly recommended that the RIC be run on its own machine.

Operating Systems: Windows Server 2008 R2 SP1

System Requirements:

- .NET 4.0 Runtime
- Processor: 2.5 Ghz or better
- Memory: 2GB RAM minimum or more. (We recommend 8GB of RAM for any of the OS's above).
- Disk Space: 200 MB for application and temporary data storage *This amount should suffice for most standard applications, however, it may vary widely depending on the type and quantity of data transferred.

Other: The service should be set to run with system access. Additionally, the assigned account should also have permissions to access network file shares should non-local file access be required.

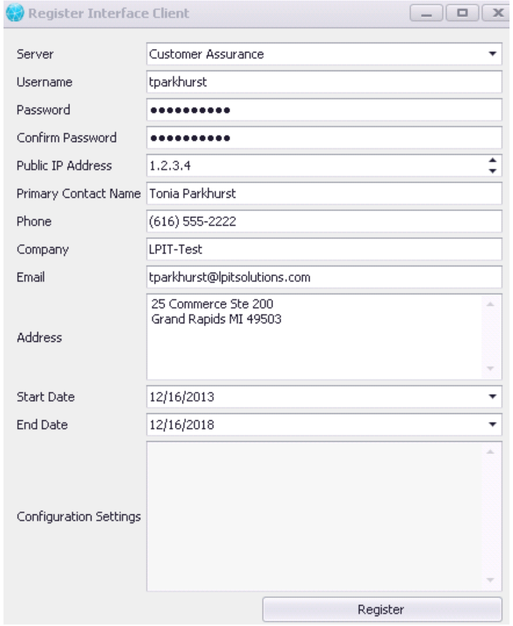
3.2.2 Installation Directions

Contact the TrackCore support team for download access to the Windows installer package for the RIC. Copy the package to the host machine and run the installer directly on the host computer. Follow the directions on screen for installing the package. Run the Configurator application in order to setup the client for communication with TrackCore including how to register the client. Details for registering the client are provided in the next section.

3.2.3 Registering the Client

1. To register a new client, extract the RegisterClient.zip file and double click RegisterInterfaceClient.exe.
2. Fill out the form displayed :
    a. Server options are Customer Assurance (test) and Production. [1st time through = Customer Assurance, 2nd time through = Production]
    b. User name and password are for use in this application and not associated with anything else.
    c. Enter the public IP address as determined during the prerequisites.
    d. Contact information should be for a key contact person in case we have issues with the feed. This can be a technical support person/team or a key user who "owns" the TrackCore application and would be able to put us in touch with a technical support person/team in case the need arose.
    e. For start date, use today.

f. For end date, use today's date plus 5 years.

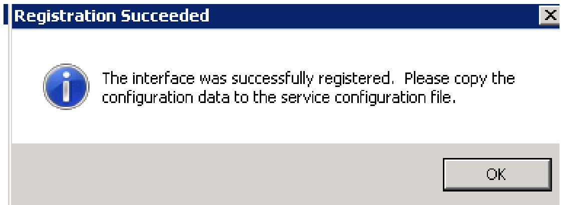


Register Interface Client

Server	Customer Assurance
Username	tparkhurst
Password	●●●●●●●●●●
Confirm Password	●●●●●●●●●●
Public IP Address	1.2.3.4
Primary Contact Name	Tonia Parkhurst
Phone	(616) 555-2222
Company	LPIT-Test
Email	tparkhurst@lpitsolutions.com
Address	25 Commerce Ste 200 Grand Rapids MI 49503
Start Date	12/16/2013
End Date	12/16/2018
Configuration Settings	

g. Select Register

i. If you receive a successful registration the Configuration Settings box will fill with html code.

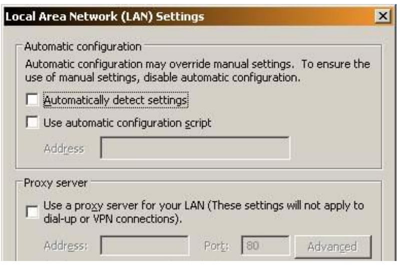


Registration Succeeded

The interface was successfully registered. Please copy the configuration data to the service configuration file.

ii. If not, look for and fix any fields with a red X next to them and try to register again.

iii. If you have continued failures registering, verify that you don't have browser security settings set too high or a LAN setting using a proxy server which would be preventing the registration.



Local Area Network (LAN) Settings

Automatic configuration

Automatic configuration may override manual settings. To ensure the use of manual settings, disable automatic configuration.

☐ Automatically detect settings

☐ Use automatic configuration script

Address

Proxy server

☐ Use a proxy server for your LAN (These settings will not apply to dial-up or VPN connections).

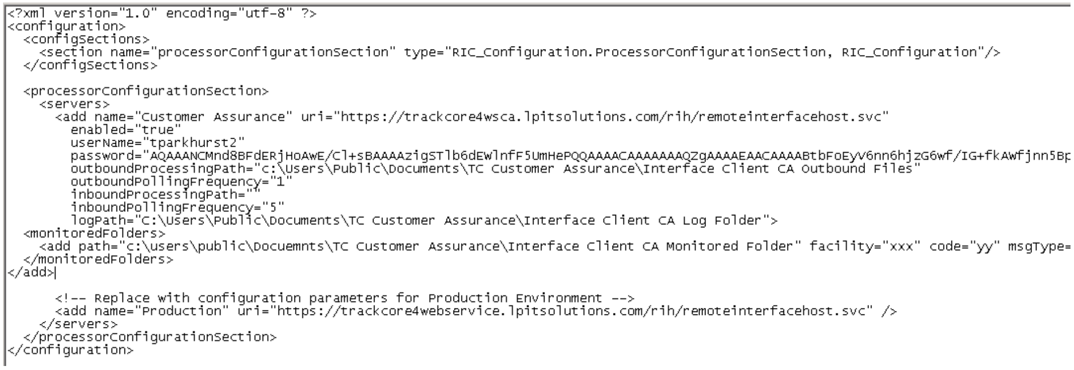
Address: Port: 80

1. Click in the Configurations Settings box on the bottom of the form once it is filled in, Hit Cntrl+A to select all, then right click and choose copy. This will copy the configuration data to the clipboard for use in the following steps.

<u>Configure Customer Assurance Application</u>

2. Next you'll update the configuration file. Open Windows Explorer and type in %PROGRAMDATA%\TrackCore\TrackCore Remote Interface and hit Enter. Open the config.xml file in Notepad.
3. Highlight the entire **<add name=…./>** tag that corresponds to the server you selected when creating the registration [Customer Assurance for 1st iteration] and Right click-Paste to replace it with the information you copied out of the registration tool.

NOTE: you may add carriage returns to the file between values to make it more readable. For example:



```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="processorConfigurationSection" type="RIC_Configuration.ProcessorConfigurationSection, RIC_Configuration"/>
  </configSections>

  <processorConfigurationSection>
    <servers>
      <add name="Customer Assurance" uri="https://trackcore4wsca.lpitsolutions.com/rih/remoteinterfacehost.svc"
        enabled="true"
        userName="tparkhurst2"
        password="AQAAANCMnd8BFdERjHoAwE/Cl+sBAAAAzigSTlb6dEwlnfF5UmHePQQAAAACAAAAAAAQZgAAAAEAACAAAABtbFoEyV6nn6hjzG6wf/IG+fkAWfjnn5Bp
        outboundProcessingPath="c:\Users\Public\Documents\TC Customer Assurance\Interface Client CA Outbound Files"
        outboundPollingFrequency="1"
        inboundProcessingPath=""
        inboundPollingFrequency="5"
        logPath="C:\Users\Public\Documents\TC Customer Assurance\Interface Client CA Log Folder">
  <monitoredFolders>
    <add path="c:\users\public\Docuemnts\TC Customer Assurance\Interface Client CA Monitored Folder" facility="xxx" code="yy" msgType=
  </monitoredFolders>
</add>

      <!-- Replace with configuration parameters for Production Environment -->
      <add name="Production" uri="https://trackcore4webservice.lpitsolutions.com/rih/remoteinterfacehost.svc" />
    </servers>
  </processorConfigurationSection>
</configuration>
```

4. Change the value of **enabled**=“false” to “true”.
5. Set the **outboundProcessingPath=“”** to the folder compressed and encrypted files will be stored by the service until they are sent. Default- “C:\Users\Public\Public Documents\TC Customer Assurance\Interface Client CA Outbound Files”
6. Adjust the **outboundPollingFrequency=”5” to “1”** if desired. The value represents minutes.
7. The **inboundProcessingPath=“”** and **inboundPollingFrequency=“5”** can be left alone.
8. Set the **logPath=“”** to the folder for log files. Default – “C:\Users\Public\Public Documents\TC Customer Assurance\Interface Client CA Log Folder”

If using monitored folder rather than port:

9. Modify the initial **<add />** line of the **<monitoredFolders />** section
    h. Add path is the folder the service will monitor. Default: add path=”C:\Users\Public\Public Documents\TC Customer Assurance\Interface Client CA Monitored Folder” (This is the location where your incoming files will be put for TrackCore Remote Interface service to encrypt, compress and move to the Outbound Processing Path Folder)
    i. facility # should be “” – will be supplied by TrackCore Support
    j. code # should be “”– will be supplied by TrackCore Support
    k. message type should be “ZIL”
    l. event type should be “ECI”
10. Add additional **<add />** lines for each additional folder to monitor. (Additional folders are necessary if there are multiple instances of facility ID, code ID, message type or event type. In most cases there will be only 1 monitored folder.)
3. 11. Save the config.xml file and close the Register Interface Client Form. Keep the config.xml file open.
4. Double click on RegisterInterfaceClient.exe to reopen as a blank form.
5. Repeat process above to Register and Configure the Production Server
6. Once config.xml is updated with both Customer Assurance and Production information, save and close the config.xml file.
7. Close the Register Interface Client form.

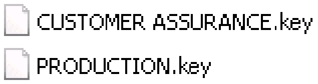
If using monitored port rather than folder:

1. At the end of the logPath entry but before the > hit enter to add a line
2. Type monitoredPort **“”** and allow line to end with the >.
3. Fill in the port which needs to be monitored for HL7 messages in between the **“”** following monitoredPort.
4. Delete the **<add />** line of the **<monitoredFolders />** section.
5. Save the config.xml file and close the Register Interface Client Form. Keep the config.xml file open.

<u>Start TrackCore Remote Interface service</u>

1. Open Windows services (services.msc) and locate the *TrackCore Remote Interface* service.
2. Right click the service and choose Start from the popup menu. Monitor for a couple minutes to verify the service stays running.
3. Notify TrackCore Customer Support that registrations have been submitted.

4. Once TrackCore notifies you that your registrations were accepted, go to %PROGRAMDATA%\TrackCore\TrackCore Remote Interface and verify that 2 keys were generated:



CUSTOMER ASSURANCE.key

PRODUCTION.key

5. For monitored folders implementations: Provide the Monitored Folders file paths to the service/program sending files to TrackCore Remote Interface for handling.
6. For monitored port implementations: Make sure HL7 messages are configured so that MSH.4 (Facility ID) and MSH.8 (Integration Code) are set to the values supplied by TrackCore Customer Support

3.3 USING RIC TO TRANSFER FILES

When configuring the RIC, multiple drop folders can be specified for the client to monitor. These folders can be associated with a specific message type or mapped to a specific destination site in TrackCore. For example multiple destinations would be specified for separate test and production TrackCore sites.

3.4 COMMUNICATING PORT TO PORT USING TCP/IP

During configuration of the RIC, the client is configured to listen to a specific TCP/IP port on the host machine. This port should not be monitored by any other applications. The sending application or interface will use standard HL7 port to port communication to send an HL7 message (or file) to the client. The RIC will respond with an HL7 ACK message indicating success. Note that this success indicator only indicates that the client successfully received the message for processing. There is no secondary success message that the message was transmitted to the end destination or that it was successfully inserted into the remote database.

The RIC can be configured to listen to separate ports for testing and production, or for other services. This requires a separate configuration for each additional port.

Messages outbound from TrackCore to a hospital system will be sent to a configured IP and Port number using standard HL7 configurations.