COMMONWEALTH OF PENNSYLVANIA
DEPARTMENT OF Human Services

INFORMATION TECHNOLOGY STANDARD

Name Of Standard: **Open TI .Net Interfaces**	Number: **STD-EASS005**
Domain: **Integration & Middleware**	Category:
Date Issued: **05/01/2009**	Issued By Direction Of:
Date Reviewed: **01/11/2011**	Clifton Van Scyoc, Dir of Div of Tech Engineering

Abstract:

The purpose of this document is to define the interface between the Windows OpenTI component and the mainframe.

This document builds upon the existing OpenTI Standards documentation. The original standards, entitled "DHS OpenTI Component Standards", were written for earlier versions of OpenTI and use with Visual Basic 6 (VB6). During this time the evolution of Middleware implementation has refined the use of OpenTI development and deployment. This addendum explains and documents this evolution for preservation and standardization of use.

This document provides enough information so that the reader can have a basic understanding of VB.NET OpenTI development. Detailed information regarding naming conventions, required forms to submit, technology overview, architectural descriptions, and other procedures can be found in the documents at the BIS website:
http://mydhs/cs/groups/webcontent/documents/document/p_032073.doc

General:

What is "OpenTI"

Unisys has produced a product called "Distributed Transaction Integration". OpenTI is a component of that. It allows many platforms to access the Unisys ClearPath mainframe. One of these interfaces is in COM+ which allows a Windows application to access the mainframe like any other data source. Distributed Transaction Integration comes with "Adapters" to talk to other software, platforms, and transaction environments such as BizTalk, BEA TUXEDO, and Fujitsu TPMS.

OpenTI can be used to have the mainframe access Windows or Windows access the mainframe.

The scope of this document is the use of OpenTI.NET with VB.NET.

Standard:

Overview

A high level description of the evolution of middleware development to the current standards and the value obtained from it.

The goals of the new standards are to:

- Improve performance
- Simplify maintenance on both the mainframe and Windows side middleware.
- Provide standardized methodologies of software use.

Anachronisms and Glossary

A glossary is at the end of the document that will explain many of the terms and abbreviations used in this document. It is not intended as a complete detailed technical reference – but rather a general background to assist reading this document.

Executive Summary

DHS began using OpenTI about 2001. This was initiated with interdisciplinary teams meeting on a weekly basis to develop forms, practices, identify responsibilities and what department they fell under, documentation, and architecture. Implementations, such as COMPASS and MCI, followed with demands to create more complicated and larger data transfers. During this time we introduced the use of XML for communication between Windows components and improved the OpenTI interface for communication with the mainframe.

The first interfaces were less efficient to code and implement. This made deployments more difficult and more likely to fail and also increased development time. The OpenTI interfaces between Windows and the mainframe went through evolutionary steps.

- Parameterized data fields in tightly bound calls with X_Common buffers
- Flexible “View” buffers that were less bound with X_Common buffers
- A single loosely bound X_Octet buffer

These changes greatly reduced the developer and administrative overhead of creating and maintaining the OpenTI interfaces. There were also improvements in efficiency as OpenTI was able to reduce and finally eliminate the amount of encoding and decoding of the OpenTI buffers with “Transparent Data”.

Many of the interfaces are very large, the original idea of tightly bound parameterized buffers using the Occurs clause was never made to work and appears not to be possible. If it can be done, the developer and machine overhead would not be worth it anyway.

The middleware code is to be encapsulated in a separately deployed application to expose it’s functionality in a reusable manner and promote ease of maintenance.

The current practice is to use a single X_Octet buffer. This should be added to the standards which are two X_Common views.

Detail

OpenTI provides many ways to create the interface between the mainframe and Windows. They leave the choice to the developers and architects of the enterprise to decide which interface is best for the enterprise. The enterprise then builds their standards to fit. Three of these interfaces will be described, and in the order that their use evolved when components were being created by the middleware group – first known as ATCS currently designated as "MSU". It should be noted that some of these features and capabilities are also available in the VB6 version of OpenTI. They became available, or were discovered, while the section had changed to coding with VB.NET and have thus been associated with .NET use.

First a general overview: Unisys has software on the OpenTI servers that the developer uses to make DLLs that define the interfaces. The DLL exposes a function call that takes information in and returns data along with the function value. There are wizards that prompt the developer to input the information required to make the interface. This is information regarding services on both the Windows and mainframe side. Some of essential information defined is:

- The name of the service in Windows
- The name of the service on the mainframe
- The data definition of the data buffer being passed
- Optionally, the host name (IKEC or HSHA at DHS)
- Parameters for error information
- The use of X_Octet or X_Common buffers
- The function has a Boolean return value, true for success

The OpenTI Builder for .NET is used to create the interface DLL. The builder is what sends up the wizards that prompt the developer for the needed information. Regardless of the choices above, you use the same builder and it creates a DLL.

On the mainframe side, the mainframe developer creates a "Proc" that defines the data. This is a hierarchal structure and defines data types, size, and the relative position of each data attribute. This relative position is extremely important. As example, let us take a persons name. The first name may be an 11 space string, followed by a one space middle initial, then a 15 space last name. This is now a string of 32 characters in length. If the value of first name is "FRED", then 7 blank spaces will need to be added to the field. Inversely, if you were to receive a string of data from the mainframe, you could extract the correct values by using the information in the proc. Incorrectly built strings will shift the information and make an error in the extraction. The proc is also used by the VB.NET developer to build their applications for this reason.

When an application needs to send or receive data to another application, regardless of the type of machine, it must put that data in a buffer. For the receiving application to use that data it must know how that data is structured. OpenTI/OLTP uses Open Group compliant buffers and user defined buffer subtypes to define the data structure. The buffer subtype specifies exactly how that data is structured. This data is structured or tailored to meet the needs of the application. A properly structured data subtype is called a "View", which is also classified as an X_Common buffer. Views will be important with one of the ways that the OpenTI Builder can create an interface.

Another type buffer that is also used by OpenTI/OLTP is the X_Octet buffer. This type of buffer is not tightly bound and is used by application programs that need to define the structure of the data to be exchanged.

X_Octet buffers are more efficient as they do not require OpenTI to interrogate the contents. This is also called transparent data and there is no encode or decode of the buffers.

All the data being passed between the Windows and ClearPath OS2200 servers will need to have a 32 to 36 bit transformation. This is done very fast by the mainframe. The encoding and decoding of the X_Common data is an additional step that would be required.

Parameters with X_Common

This is how we built the first OpenTI applications. The function call has parameters for each attribute of data. Thus – if you were going to send a record set to the mainframe – each value in the record set would have to be mapped to the appropriate parameter of the function call. These parameters would need to be mapped for both input and output information.

An example of a simple strongly typed function call would be:

```
'
'   Invoke the method call in the OpenTI object
'
lngReturnStatus = objOpenTI.AIM02SVC01 _
( _
    strSSNIn, _
    strEligibilityStatus, _
    strInqRslt, strSSNOut, _
    strRecipientNbr, strLineNbr, _
    strCashCounty, strCashRecord, _
    strCashCategory, strFSCounty, _
    strFSRecord, strFSCategory, _
    strCaseload, strDistrict, _
    strFirstName, strLastName, _
    strMidInit, strApartment, _
    strAddress1, strAddress2, _
    strCity, strState, _
    strZipCode, strZipExt, _
    strPhoneNbr, strDOB, _
    strCashMedElig, strFSEligibility, _
    strProjectNbr, _
    lngTPurCode _
)
```

The above call inputs a single parameter of Social Security Number (strSSNIn) and then 29 parameters of data (two per line) are returned from the mainframe. The return of the function call is an error code of lngReturnStatus. One of the parameters is lngTPURCode and is also error information.

This creates a tightly bound interface that models the data and must be maintained. Mattering on the data model, this interface can start to become quite complex. Use of X_Common parameters also led to concerns with the looping of data, known as “Occurs”. Occurs are often nested and development of X_Common parameterized interfaces with this is difficult, and to date, not proven.

There is a technical reason for the problem with the Occurs structure in a parameterized X_Common function call. This X_Common typed buffer is a user-defined buffer, with its unique data structure, is created as a source element using an editor such as IPF. Cobol restricts data types structure descriptors to

long int, short int, and char. When the Vadd tool is used to create and install the view into the OLTP Configuration buffer definition file on the Mainframe, it will convert the source element into a Cobol copy-book (proc) to be utilized by the applications. Those source data types are converted to comp or comp-5 signed numeric data structures. The char source data definitions are converted into alphanumeric data types within the proc. Those numeric and alpha numeric fields are assigned to a 05 elementary level. The only data structures that will accommodate an Occurs is the int (short or long). There is no provision for establishing a subordinate Occurs to the 05 elementary level data item. Also there is no provision for establishing an Occurs for an alphanumeric data item.

The code in the function call is also more difficult to read and maintain. The only way to tell the difference between the input and output data is by naming conventions and how familiar the developer is with the project.

If a parameter is added, subtracted, or data type changed the interface is broken and a new DLL will need to be made.

The purpose of a function call is to send and receive data. Sub procedures should be used for the logic involving the use of nested occurs and any conversion of data types. Instead this logic is now inserted into the function call.

The limitation of tightly bound parameter lists quickly became evident as the first interfaces were built and deployed. Even a minor change to the interface requires rebuilding the OpenTI DLL. This must be deployed to the server along with all the other new objects. The deployment operation will require removal and redeploy to the GAC and the same to the COM+ application on the server. There are also detailed changes that must be made on the mainframe side. This also creates the need for versioning of these DLLs to specific deployments.

Sized Views with X_Common

The limitation of tightly bound parameter lists set the middleware developers looking for another solution. It was at this time the use of views was proposed and the value quickly proven in development.

A “View” is a text file generated by the mainframe developer and a copy sent to the Windows VB.NET developer. It has a “.v” file extension. The purpose of the view is to define the size of a buffer. There is both an input and output buffer. These buffers are independently sized as needed. The OpenTI builder consumes the views to create the OpenTI interface DLL. The VB.NET developer is prompted by the wizard to input the views and other information to make the DLL. It is then deployed to the servers.

What is different is that the view itself is just the size of the data string that is being sent to the mainframe. The individual attributes are not declared. This greatly decouples the interface. When the mainframe proc is established, “Filler” is added at the end. An example would be if there were 1,889 characters of data. A buffer of 111 character filler would be added at the end and a view of 2,000 created.

If attributes are added to the proc – it is in the filler. The OpenTI DLL does not know there is a difference. Entire attributes can be moved, changed, data types change, and the interface DLL will not be broken as long as the view size does not change.

There was an immediate improvement with the ease of maintenances. Interface DLLs could go years without the need for rebuilding if ever. The views are reusable and if two different methods in an application had the same size buffers then they could use the same views.

This is an example of an X_Common function call using views:

```
'
```

```
'   Invoke the method call in the OpenTI object
'
lngReturnStatus = objOpenTI.MCI05SVC01 _
    ( _
        strInputView, _
        strOutputView, _
        lngTPurCode _
    )
```

As one can see in the above, we still have the function return code and the TPurCode, but now the input and output buffers are simplified and easier to read and maintain. All complications with changing data types and making the data strings are encapsulated away from the function call.

Unsized Buffer with X_Octet

This is the technique that is currently being used for all new .NET development. There is one buffer that is both the input and out of the OpenTI call. It is not sized and entirely decouples the data model from the OpenTI interface. Simply put – one size fits all. If there is an addition or reduction of the field, the existing buffer can be used with only minor changes made on the OLTP side. The developers simply make the changes to their business objects and deploy them without having to change the OpenTI interfaces.

The experience gained over time by the middleware team has been the majority of maintenance is usually adding a field to an existing interface. In comparison to tightly bound X_Common parameters, the use of X_Octet greatly reduces the costs in development, deployment, and testing. Different projects using X_Common views and X_Octet buffers were at one point done concurrent with the preference emerging with the X_Octet buffer.

This is an example of an X_Octet function call:

```
        '
        '   Invoke the method call in the .NET OpenTI object
        '
        blnReturnType = objOpenTI.ZIG01SVC01 _
                    ( _
                        strX_OctetBuffer, _
                        intTpurCode _
                    )
        '
```

Use of X_Octet to pass environment and host information

One of the problems through this development was that the OpenTI DLLs were named and compiled to be specific to the development environments. A DLL for “Dev” was compiled for the “U” services, “SAT” for the “S” services, and production was of course done with a “P”. Though a small complexity, it did add to the developer list of things to keep track of since we had a folder full of DLLs with each migration. It was discovered that OpenTI makes it possible to pass the service name in at runtime and avoid this. The middleware was then made “Smart” to know what environment it was operating in and call the proper service at runtime. This was accomplished with no measurable loss of performance.

There are also some applications that require 24 hour access. OpenTI allows the function call to specify which mainframe host to go to at runtime. Once again the middleware is aware of which host is needed and passes that in the OpenTI call at runtime. This has been done with PROMISe for quite some time and there are other projects that desire this functionality in the future.

This is an example of an X_Octet function call that passes host and service parameters:

```
'
'   Invoke the method call in the .NET OpenTI object, Boolean
'
blnReturnType = objOpenTI.JNT01SVC01 _
        ( _
           strX_OctetBuffer, _
           strRemoteHost, _
           strService, _
           intTpurCode _
        )
```

Current Standards
Windows Developer

Windows Program Language	VB.NET
Visual Studio Version	VS2008, Framework 3.5
OpenTI Interface	X_Octet
OpenTI Service	Pass at runtime
Host	Only pass for 24/7 availability

The middleware code is to be encapsulated in a separately coded and deployed business object. Naming standards are in the original middleware standards object. It should not be part of the calling application.

Summary

OpenTI development went through several evolutionary phases as the middleware group perfected its use and gained experience. Part of this experience was with the interface DLL. At first glance, using OpenTI to describe the data with parameters seemed like a good idea but was quickly dropped. X_Common views improved the experience greatly and the adoption of X_Octet was a further improvement.

Using tightly bound X_Common parameter interfaces were problematic as:

- OpenTI didn't give you a graceful way to deal with data structures that used the COBOL "OCCURS" clause. Research to date shows that this may not even be possible with certain data types.
- There is additional encoding and decoding of those parameters.
- Even small changes to the interface required building a new OpenTI interface dll for a service, binary compatibility was broken and deployments and compiles were required.
- Most of the data interchanges because very large, making for a lot of tedious work in the OpenTI Builder.
- The large interchanges make for a difficult to maintain function call.
- Testing of the parsing is indirect. You can't step thru the parsing in the OpenTI DLL in Visual Studio. You could only see the results. This makes debugging and development harder.
- Changes also required administrative overhead in other groups at DHS.

Using X_Common Views were a good improvement

- The OpenTI buffers were built on View files required less changes as the interface was not normally broken when changes were made to the data structure of the service.
- This reduced administrative overhead as the views and interfaces that were the purview of administrators in another group.
- You can define the buffer to have some extra space, so that the data interchange can grow and change without administrative and binary compatibility issues.
- You can test the parsing by using the tools in Visual Studio such as the debugger.

Using X_Octet Buffer:

- Eliminated the encoding and decoding of the buffers.
- Eliminated the administrative overhead of keeping views.
- Eliminated recompile and redeployments of the OpenTI DLL when interface changes were made.
- Eliminated attempts to define the data structure in the function call
- Makes the function call easy to read and maintain since the parameters can be clearly shown for environment, host, data, and error codes.

Glossary

ATCS – Advanced Technologies Competency Section. This section was to learn a new technology and then aid other units and sections at DHS to learn how to use it also. In this way the section was on the edge of many different technologies than just OpenTI and would loan out a staff person to help with use and mentor other sections with that technology.

COM+ - An environment for hosting DLLs. It provides services such as transaction management, 2 Phase Commits, and monitoring.

DLL – Dynamically Linked Library. A compiled unit of code that performs a segment of work and provides that functionality. They cannot work on their own but must be hosted in an environment or memory space provided by some other application. DLLs are called by another application that needs their functionality and are thus "Instantiated".

MSU – Middleware Services Unit. The section was downgraded to unit along with many others in a DHS reorganization. The name change was in regards to the fact that most of the work done by the unit was middleware based.

Open Group – Is a "vendor-neutral and technology-neutral consortium, who's goal is to enable access to integrated information, within and among enterprises, based on open standards and global interoperability." More information is available at this link: http://www.opengroup.org/

Visual Studio – A product that provides a development environment to create solutions in languages such as VB.NET and with tools like SQL Server and BizTalk. Versions are often referred to as "VS(year), such as "VS2008" for the 2008 release.

Wizards – A series of dialog boxes that prompt for information to achieve a task.

Exemptions from this Standard:

There will be no exemptions to this standard.

Refresh Schedule:

All standards and referenced documentation identified in this standard will be subject to review and possible revision annually or upon request by the DHS Information Technology Standards Team.

Standard Revision Log:

Change Date	Version	Change Description	Author and Organization
05/01/2009	1.0	Initial Creation	R. Ziegler, MSU
01/11/2011	1.1	Reviewed content – No changes	R. Ziegler, MSU
2/25/2016	1.1	Reviewed content, changed DPW to DHS, updated a link. No other changes	R. Ziegler, MSU