SYBASE®

MSG-IDE Guide

e-Biz Impact

5.4.5

DOCUMENT ID: DC10097-01-0545-01

LAST REVISED: July 2005

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

Contents

About This Book

Audience

This book is for developers who create applications using e-Biz Impact™ version 5.4.5.

How to use this book

This guide is organized into the following chapters:

- Chapter 1, “Overview” provides a introduction to MSG-IDE basic concepts and terminology and describes MSG-IDE objects.
- Chapter 2, “Developing ODL Applications” discusses the development of e-biz Impact Object Definition Language (ODL) acquisition and delivery AIMs.
- Chapter 3, “Using MSG-IDE” describes how to use MSG-IDE to build ODL acquisition and delivery AIMs.

Related documents

e-Biz Impact documentation The following documents are available on the Sybase™ Getting Started CD in the e-Biz Impact 5.4.5 product container:

- The e-Biz Impact installation guide explains how to install the e-Biz Impact software.
- The e-Biz Impact release bulletin contains last-minute information not documented elsewhere.

e-Biz Impact online documentation The following e-Biz Impact documents are available in PDF and DynaText format on the e-Biz Impact 5.4.5 SyBooks CD:

- The *e-Biz Impact Application Guide* provides information about the different types of applications you create and use in an e-Biz Impact implementation.
- The *e-Biz Impact Authorization Guide* explains how to configure e-Biz Impact security.
- *e-Biz Impact Command Line Tools* describes how to execute e-Biz Impact functionality from a command line.
- The *e-Biz Impact Configurator Guide* explains how to configure e-Biz Impact using the Configurator.

- The *e-Biz Impact Feature Guide* describes new features, documentation updates, and fixed bugs in this version of e-Biz Impact.
- The *e-Biz Impact Getting Started Guide* provides information to help you quickly become familiar with e-Biz Impact.
- The *Monitoring e-Biz Impact* explains how to use the Global Console, the Event Monitor, and alerts to monitor e-Biz Impact transactions and events. It also describes how e-Biz Impact uses the standard Simple Network Management Protocol (SNMP).
- *Java Support in e-Biz Impact* describes the Java support available in e-Biz Impact 5.4.5.
- The *e-Biz Impact MSG-IDE Guide* (this book) describes MSG-IDE terminology and explains basic concepts that are used to build Object Definition Language (ODL) applications.
- The *e-Biz Impact ODL Guide* provides a reference to Object Definition Language (ODL) functions and objects. ODL is a high-level programming language that lets the developer further customize programs created with the IDE tools.
- The *e-Biz Impact TRAN-IDE Guide* describes how to use the TRAN-IDE tool to build e-Biz Impact production objects, which define incoming data and the output transactions produced from that data.

Note The *e-Biz Impact ODL Application Guide* has been incorporated into the *e-Biz Impact ODL Guide*.

The *e-Biz Impact Alerts Guide*, the *e-Biz Impact SNMP Guide*, and the *e-Biz Impact Global Console Guide* have been combined into a new guide—*Monitoring e-Biz Impact*.

Adaptive Server Anywhere documentation The e-Biz Impact installation includes Adaptive Server® Anywhere, which is used to set up a Data Source Name (DSN) used with e-Biz Impact security and authorization. To reference Adaptive Server Anywhere documentation, go to the Sybase Product Manuals Web site at Product Manuals at http://www.sybase.com/support/manuals/, select SQL Anywhere Studio from the product drop-down list, and click Go.

Note Read the *SyBooks Installation Guide* on the Getting Started CD, or the *README.txt* file on the SyBooks CD for instructions on installing and starting SyBooks.

Other sources of information

Use the Sybase Getting Started CD, the SyBooks CD, and the Sybase Product Manuals Web site to learn more about your product:

- The Getting Started CD contains release bulletins and installation guides in PDF format, and may also contain other documents or updated information not included on the SyBooks CD. It is included with your software. To read or print documents on the Getting Started CD, you need Adobe Acrobat Reader, which you can download at no charge from the Adobe Web site using a link provided on the CD.
- The SyBooks CD contains product manuals and is included with your software. The Eclipse-based SyBooks browser allows you to access the manuals in an easy-to-use, HTML-based format.

  Some documentation may be provided in PDF format, which you can access through the PDF directory on the SyBooks CD. To read or print the PDF files, you need Adobe Acrobat Reader.

  Refer to the *SyBooks Installation Guide* on the Getting Started CD, or the *README.txt* file on the SyBooks CD for instructions on installing and starting SyBooks.
- The Sybase Product Manuals Web site is an online version of the SyBooks CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

  To access the Sybase Product Manuals Web site, go to Product Manuals at http://www.sybase.com/support/manuals/.

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

❖ **Finding the latest information on product certifications**

1. Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.
2. Select Products from the navigation bar on the left.
3. Select a product name from the product list and click Go.
4. Select the Certification Report filter, specify a time frame, and click Go.
5. Click a Certification Report title to display the report.

❖ **Creating a personalized view of the Sybase Web site (including support pages)**

Set up a MySybase profile. MySybase is a free service that allows you to create a personalized view of Sybase Web pages.

1 Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.

2 Click MySybase and create a MySybase profile.

Sybase EBFs and software maintenance

❖ **Finding the latest information on EBFs and software maintenance**

1 Point your Web browser to the Sybase Support Page at http://www.sybase.com/support.

2 Select EBFs/Maintenance. Enter user name and password information, if prompted (for existing Web accounts) or create a new account (a free service).

3 Select a product.

4 Specify a time frame and click Go.

5 Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

Conventions

The syntax conventions used in this manual are:

Key	Definition
commands and methods	Command names, command option names, utility names, utility flags, Java methods/classes/packages, and other keywords are in lowercase Arial font.
variable	Italic font indicates: • Program variables, such as *myServer* • Parts of input text that must be substituted, for example: *Server*.log • File names
File | Save	Menu names and menu items are displayed in plain text. The vertical bar shows you how to navigate menu selections. For example, File | Save indicates "select Save from the File menu."

Key	Definition
`package 1`	Monospace font indicates: • Information that you enter in a graphical user interface, at a command line, or as program text • Sample program fragments • Sample output fragments

Accessibility features

This document is available in an HTML version that is specialized for accessibility. You can navigate the HTML with an adaptive technology such as a screen reader, or view it with a screen enlarger.

For information about how Sybase supports accessibility, see Sybase Accessibility at http://www.sybase.com/accessibility. The Sybase Accessibility site includes links to information on Section 508 and W3C standards.

If you need help

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

CHAPTER 1

Overview

Application Interface Module (AIM) applications are the e-Biz Impact components that acquire data from and deliver data to endpoint applications.

This chapter describes how AIM applications function and provides a summary of the creation process using the e-Biz Impact MSG-IDE tool.

Note AIMs are developed using the Object Definition Language (ODL). See the *e-Biz Impact ODL Guide* for more information on ODL.

Introduction

e-Biz Impact uses Application Interface Module (AIM) applications to communicate with other systems, acquire transaction data from an endpoint application (a logical or physical device), and deliver the data to one or several endpoint destinations.

AIMS can be developed using ODL, Java, and C\C++.

- ODL AIMs – use MSG-IDE and this guide to develop acquisition and delivery AIMs (also known as message AIMs) using the e-Biz Impact Object Definition Language (ODL).
- Java AIMs – to develop Java AIMs, see *Java Support in e-Biz Impact* on the e-Biz Impact SyBooks CD that comes with the product.

- C/C++ AIMs – AIMs that transmit data to a different cluster, and MQAcq and MQDel AIMs that interact with IBM WebSphere MQ, see the *e-Biz Impact Application Guide* on the e-Biz Impact SyBooks CD that comes with the product.

Data acquired by an AIM application can also be transformed in a variety of ways using transaction production, which is optional. The Store and Forward Manager (SFM) performs transaction production using ODL applications that you develop using the TRAN-IDE tool. See the *e-Biz Impact TRAN-IDE Guide* for instructions on creating SFM applications and specific details about transaction production.

MSG-IDE provides a means to build custom message AIMs for communicating between applications and e-Biz Impact. The resulting program can act as either an acquisition or delivery AIM.

Using MSG-IDE

MSG-IDE tool, a Windows-based tool, with a graphical user interface that allows you to easily develop ODL applications.

MSG-IDE terminology

The following terms are used in this guide to describe the creation of ODL AIMs using MSG-IDE:

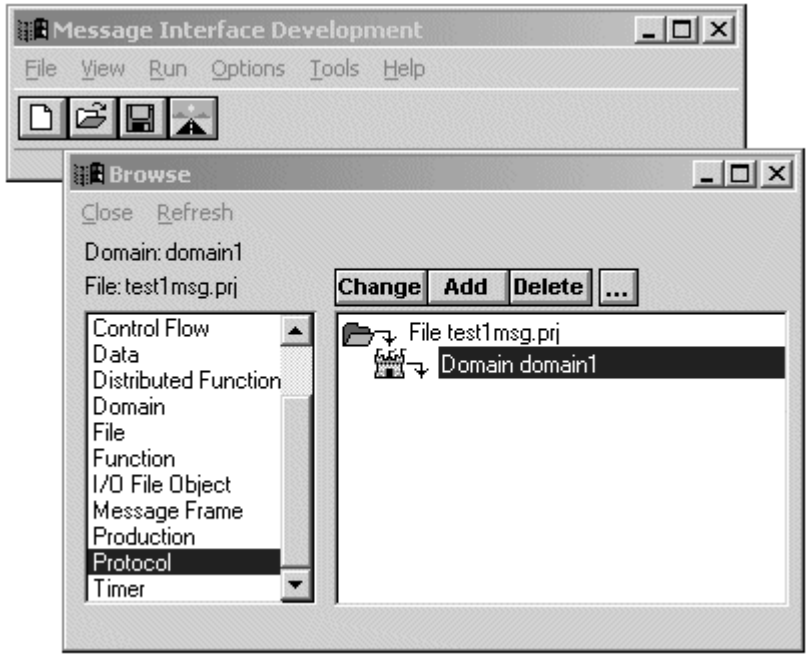
Term	Definition
File	The physical containers for objects in a message AIM. Files are saved individually with a *.fle* extension. File names have a maximum of 256 characters.
Child file	A file nested within another file. Saving a parent file does not automatically save the associated child file because the parent file contains a reference to the child file, not the actual file
Parent file	Any file that contains a child file. When files are nested more than two deep, a file can be both a parent and a child file.
Peer file	A file at the same level as another file.
Root file	Any file displayed at the first level in the objects list of the Browse dialog box.

Term	Definition
Project	A file with a *.prj* extension. This file is run as the message AIM. A project can contain only one file or multiple child files
Domain	Logical containers for the objects in a message AIM. MSG-IDE objects require a domain. A file can contain zero or multiple domains, depending on the requirements of your program; however, all objects in a project must be within the scope of a domain, even if an individual file does not contain a domain.

Starting MSG-IDE

To start MSG-IDE, select Start | Programs | Sybase | e-Biz Impact 5.4 | Msg-IDE. When the tool executes, two windows display (Figure 1-1):

Figure 1-1: MSG-IDE tool



Message Interface Development window The main MSG-IDE window allows you to create and save new applications. Applications are saved as project files (*.prj*), and you can have only one application per project.

Browse window The Browse window lists the files you currently have open, their domain, and their objects. The object list in the left pane displays the available objects. Once you create objects, the list in the right pane displays a tree view of the currently open project file and its domains, and a list of the objects that have been created within that project. For example, if you select Protocol in the left pane list, the right pane displays the names of all the protocol objects created for the currently open project You can also choose All Objects to display all existing objects in the current project.

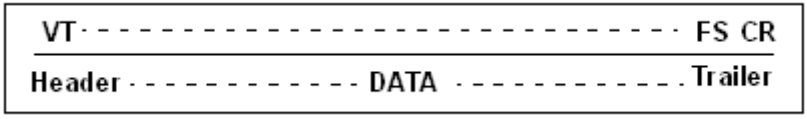
Note If the Browse window is not visible, select View | Launch Browser from the Message Interface Development window.

MSG-IDE objects

To begin building an AIM, you select an object from the object list and click Add. Table 1-1 lists each object type and its primary function.

Table 1-1: Message interaction objects

Object type	What it does
Major objects:	
Domains and files	• Domains – logical containers for acquisition and delivery AIM project (*.prj*) files. Each AIM must have a project file. • Files – in the MSG-IDE context, files are containers for MSG-IDE objects that are common to more than one project, which allows you to build objects once, save the object in a file (*.fle*), then include the file in each project that requires the common objects.
Communications	Identifies the connection component that the AIM uses to communicate with the endpoint application.

Object type	What it does
Control flow	A container of functions and data objects that control the processing of data. Defines the startup and shutdown processing for TCP/IP AIM communication object, and define the actions to take and the functions to perform once a message frame object matches on incoming data. **Note** You must always pass the argument "`blob *pb`" to a control flow object because the control flow object always receives the data in a blob. A control flow object can be called by: • A protocol object after the protocol object receives data from the communication object. • A message frame object after that message frame matches on data. • Another control flow object.
Message frame	Defines the characteristics of the data coming from the source endpoint system by describing the data. Message frame objects include the number of bytes, all data, hexidecimal, decimal, octal, ASCII, and control characters. For example, the data below uses the message frame object to match a transaction that includes VT as the header control character and FS & CR as the trailer control characters.  Objects in the message frame matching list are ordered in the sequence of the incoming message. In this example, the first item in the list would be the control character VT.
Protocol	Regulate the flow of data from the communication object to the message frame objects. It is the main logical container for other message interaction objects. **Note** All data sent to an acquisition AIM by the communication object enters the protocol object, which then controls the passage of data to the message frame objects. A protocol object must be started in the clinit() function or with the process() method. The protocol object identifies: • The communications object to use • The available message frame objects • The control flow objects to run after the communication object connects to the port and after communication object closes or losses the port connection • The control flow object to run when the protocol object receives data.

Supporting objects:

Object type	What it does
Data	Provide data variable definitions for data that you may want to manipulate or access at a later point in the program. A data object is any data variable defined with MSG-IDE or TRAN-IDE. Create data objects to build a distributed function call command for gathering or sending data, to add a symbolic name definition, or to build your own structures or class definitions.
Distributed Function	Used to build distributed function call (DFC) commands for the client and its corresponding DFC entry point for the server.When the client needs to gather or send data, to an application endpoint, it uses a DFC command to send a transaction to the server.
Function	Within a control flow, a function object defines the actions to take, either on the data itself or because of the data's presence. When not within a control flow, a function object defines the actions to take to service a DFC command. Function objects are user-provided routines within which you may perform data manipulation and actions.
Other objects:	
Blob	A blob object (Binary Large Object Block) is a special type of data object. A blob may contain any type of data, including null bytes. Blob objects have methods that you can use within a function object to manipulate the contents of the blob.
Database Interface	Allows ODL code to access external SQL databases.
I/O File Object	Allows you access disk files or to read the contents of a directory.
Production	Displays the production objects when you open a TRAN-IDE project file in MSG-IDE. You can only display production objects in MSG-IDE; you must use TRAN-IDE to edit or remove the objects.
Timer	Allows you to set an alarm to keep track of the time a user spends doing certain activities, or to execute specific commands on a periodic basis.

Creating MSG-IDE projects

A message AIM gathers protocol messages from a source and breaks the message's data out for other use. It also gathers a set of messages into a single message format and surrounds the resulting data stream with the appropriate protocol to send it to an application.

A message AIM can function as an acquisition AIM or a delivery AIM. The structure of a message AIM and the object flow are different depending on whether the message AIM is an acquisition or delivery AIM and whether the message AIM uses a protocol object.

To create and implement ODL AIMs:

1 Use MSG-IDE to create acquisition and delivery AIMs and save them as project files. See Chapter 2, “Developing ODL Applications.”

2 Use the Configurator to specify acquisition and delivery AIM properties such as communication protocol (resources) and functions. See the *e-Biz Impact Configurator Guide*.

3 Deploy the MSG-IDE project files to the network server. See the *e-Biz Impact Configuration Guide*, Chapter 5, “Deploying Files and Executing e-Biz Impact Clusters.”

Project organization

When you create an acquisition or delivery AIM, you populate it with message interaction objects, then save the application in a project file with a *.prj* filename extension. The project file is the file that runs the AIM.

You can organize a project in any of three ways:

1 Have one MSG-IDE project file (*.prj*) and domain that contains all the objects needed for each AIM. For a simple e-Biz Impact implementation, you would have a project file for the acquisition AIM and a project file for the delivery AIM. This is the simplest organization choice, and should be used whenever possible.

2 Have one MSG-IDE project file and multiple domains that contain the objects needed for the AIM. Organize the project in this manner when you require some objects to be static and other objects to be global.

3 Have a MSG-IDE project file with one domain and several child files (*.fle*), with or without domains, that contain the objects needed for the AIM. Use this method when you have data objects, functions, or other objects that are common to more than one project. This allows you to build the objects once, in a child file, then include the file in each project that requires the common objects.

The components of a message AIM are called message interaction objects. These objects interact with a communication process to gather data from and send data to endpoint applications. Message interaction objects include protocol, communications, message frame, and control flow objects. Supporting objects include data and functions.

e-Biz Impact provide the MSG-IDE tool To develop ODL acquisition and delivery AIMs, use the e-Biz Impact provided

Rules and restrictions

When you create acquisition and delivery AIMs, keep in mind:

- For each e-Biz Impact implementation, you must develop at least one acquisition AIM and one delivery AIM.
- Each AIM must have an associated project file with a *.prj* file name extension. The project file can contain all objects required by the AIM or some of the objects and child files that contain the remainder of the objects.
- All AIM objects must be within the scope of a domain, and the project file must have at least one domain. If a project's objects are outside the scope of a domain, the AIM does not run.
- Duplicate domain names prevent the execution of an ODL project.
- All acquisition AIMs must use a clinit and a cldeinit function.
- All delivery AIMs must use a clinit, a cldeinit, a servayt, and a servproc function.

Building acquisition AIMs

An acquisition AIM obtains data from a source, packages the data into a transaction, and sends it to the SFM via a distributed function command (DFC). The input source can be a data file generated by another application or an application on an endpoint system.

Figure 1-2: Acquisition AIM flow

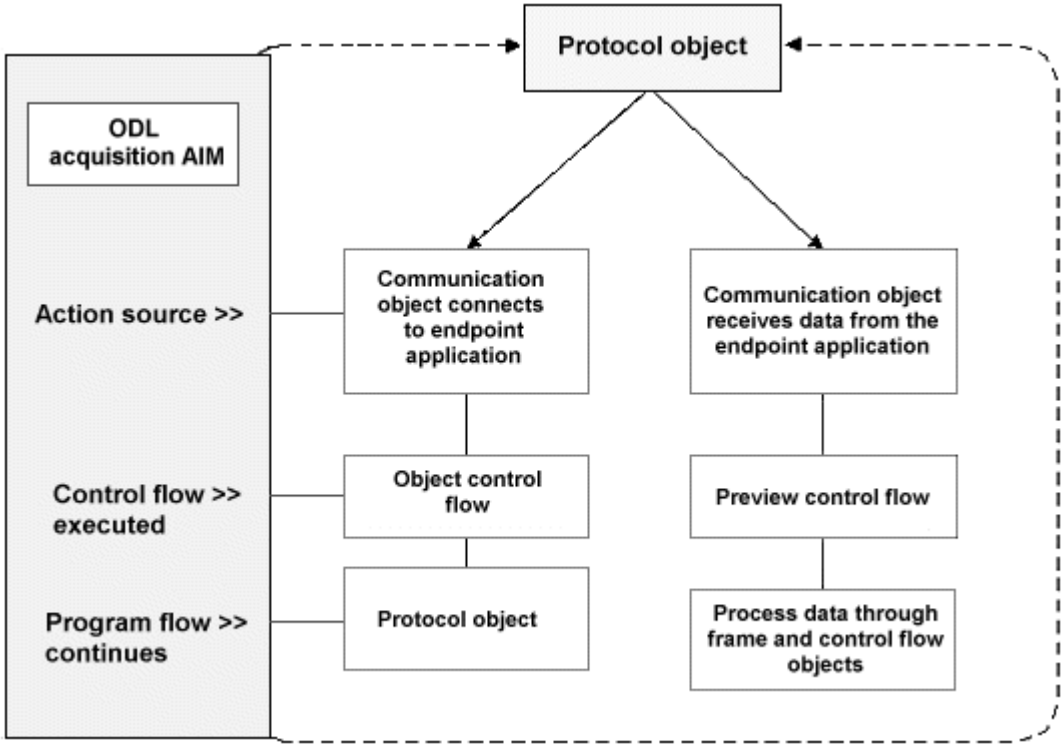


Figure 1-2 illustrates this ODL acquisition AIMs object flow:

1 When the communication object connects to the designated port, it notifies the protocol object of a successful connection.

2 The protocol object executes the control flow object listed in its Open box and waits to receive data from the communication object.

3 Once data is received, the protocol object executes the control flow object listed in the Preview box.

4 The protocol object processes data through the defined message frame and control flow objects, then waits to receive more data from the communication object.

5 If the communication object looses the connection to the endpoint, the protocol object executes the close flow.

See Chapter 3, “Using MSG-IDE,” for instructions.

Acquisition AIM structure

- Using a communication object, a data packet is received from the endpoint application.
- Headers/footers and other protocol characters are stripped using message frame objects and the data is placed into a blob variable.
- If necessary, the control flow attached to the message frame performs data blob manipulation. The AIM may contain multiple message frames with their associated control flows.
- The AIM receives the next packet of data from the communication object and processes it through message frames.
- The framed data is appended to the *blob* variable.
- The message frame object and control flow continue the append action until the *blob* data variable accumulates the transaction data, then sends the data to an SFM by calling one of the SFM routing functions (route_vprod(), route_vrec(), and so on). See the *e-Biz Impact TRAN-IDE* Guide for information on routing functions.
- An acknowledgement is sent to the endpoint based on the return value from the SFM using the send() method.

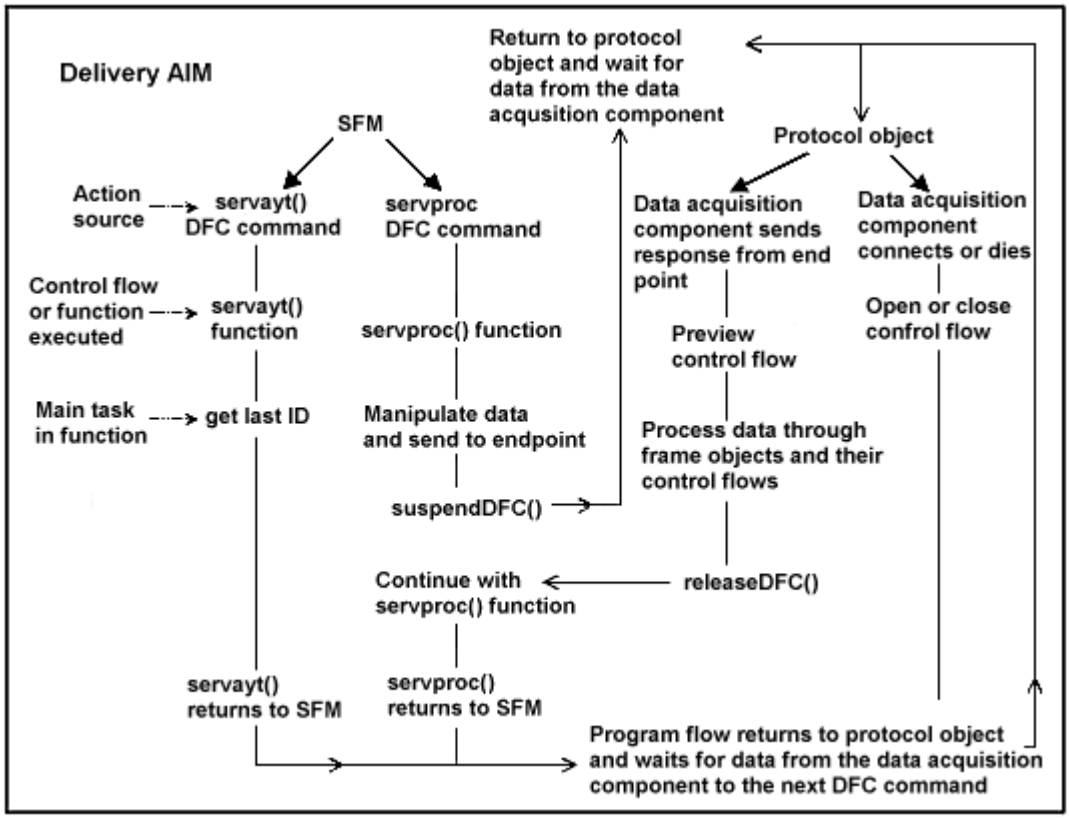
  Sources defined in the SFM configuration can receive a ping() DFC. If you do not have a ping() DFC entry point in your acquisition AIM configured to receive these commands, unpredictable results may occur.

Building delivery AIMs

A delivery AIM receives transactions routed by an SFM and sends them to the endpoint application.

Program flow in a delivery AIM can begin either with a function that services a DFC or with the protocol object. Figure 1-3 illustrates the program flow through a delivery AIM, and the action sources that trigger entry into the AIM.

Figure 1-3: Delivery AIM flow



1 An SFM sends a servayt() DFC. The message AIM executes the servayt() function to service the DFC. This function gets the serial number of the last transaction that the AIM processed from the *lastid* file. The function finishes by sending that serial number to the SFM. Program control returns to the protocol object, which waits to receive data from the communication object, or waits for the SFM to send another DFC.

2 The SFM sends a servproc() DFC. The message AIM executes the servproc() function to service the DFC. This main task of this function is to put the data received from the SFM into the format required by the endpoint application and send the data through the communication object to the endpoint.

After servproc() sends the data to the endpoint, it calls clSuspend() to suspend processing of servproc() to wait for a response from the endpoint application. When the endpoint sends a response, the protocol object processes the response data through the preview control flow, then through the message frame objects and their control flows.

After the endpoint's response is fully processed, the final control flow runs on the response data and calls clRelease() to return processing control to servproc(). The servproc() function finishes any required processing, including writing the transaction's serial number to the *lastid* file and returning the appropriate value back to the SFM. The appropriate value sent back to the SFM may be an error condition if the endpoint system did not accept the transaction.

The AIM waits for the next DFC from the SFM. Program control returns to the protocol object, which waits to receive data from the communication object, or waits for the SFM to send another DFC.

3 The communication object connects or dies. When the communication object connects to its designated port, it notifies the protocol object that it connected. The protocol object executes the control flow object listed in its Open box. Program control returns to the protocol object, which waits to receive data from the communication object, or waits for the SFM to send a DFC.

If the communication object terminates unexpectedly, the protocol object executes the control flow object listed in its Close box. In this case, it uses the restart() method in the close control flow to restart the communication object or uses a timer object to delay restarting the connection.

4 The communication object gets a response from the endpoint. After servproc() sends data through the communication object to the endpoint, it calls clSuspend() and waits for a response from the endpoint. When the protocol object receives a response about the transaction from the endpoint application, it executes the control flow object listed in its Preview box. The protocol object processes the data received from the endpoint application through its message frame objects and their control flows.

After the endpoint's response is fully processed, the final control flow runs on the response data calls clRelease() to return processing control to the servproc() function. If the protocol object receives data through the communication object while the AIM is processing a DFC, the data queues at the protocol object until the function processing the DFC calls clSuspend().

The communication object does not have to be connected to its port before the AIM can begin servicing a servayt() or servproc() DFC from the SFM. However, the communication object must be connected before the servproc() function can send the data to the endpoint application.

Delivery AIM structure

- The AIM receives servayt() command from the SFM.
- It executes servayt() function. The function reads the *lastid* file to retrieve the serial number of the last transaction processed by the AIM. The serial number is then returned back to SFM.
- The AIM receives servproc() DFC from the SFM
- It executes servproc() function. The function may perform necessary manipulation to put data in the format required by the endpoint application. The data then is sent to the endpoint using communication object's send() method. At this point, servproc() calls clSuspend() to pass control to the protocol object to process the endpoint acknowledgement.
- The protocol object executes preview control flow when the communication object receives the response from the endpoint.
- The endpoint response is processed through defined message frame objects to determine if the endpoint accepted or rejected the transaction. The control flow attached to the message frame calls clRelease() to return control to the servproc() function.
- If the endpoint accepts the transaction, servproc() saves the serial number to the *lastid* file.
- If the transaction is executed successfully, servproc() returns appropriate value to the SFM.

CHAPTER 2

Developing ODL Applications

This chapter provides a summary of the concepts and procedures used to build e-Biz Impact Object Definition Language (ODL) acquisition and delivery AIMs.

Introduction

An acquisition AIM acquires data from an endpoint application through a direct data feed over a communication connection (such as TCP/IP, TTY, or Telnet) or by reading data from one or more files, and sends the information to an SFM for optional transaction production.

A delivery AIM receives transactions from an SFM and sends them to the destination endpoint.

To develop acquisition and delivery AIMs, follow these steps:

1 Use MSG-IDE to create the AIMs. See Chapter 3, “Using MSG-IDE.”

2 Use the Configurator to specify acquisition and delivery AIM properties such as a communication protocol (resources) and routing functions. See the *e-Biz Impact Configuration Guide*, Chapter 4, “Configuring Applications,” for detailed instructions.

3 Deploy the MSG-IDE project files to the location of the e-Biz Impact server. See the *e-Biz Impact Configuration Guide*, Chapter 5, “Deploying Files and Executing e-Biz Impact Clusters.”

In addition to developing acquisition and delivery AIMs, you also must develop any SFM objects necessary for transaction production (see the *e-Biz Impact TRAN-IDE Guide*), and implement optional security authorization (see the *e-Biz Impact Authorization Guide*). See the *e-Biz Impact Getting Started Guide* for an overview of all the procedures that are necessary to develop and configure an e-Biz Impact implementation.

Building ODL acquisition AIMs

The program flow of an acquisition AIM is dependent on its functionality, but in general, the application executes as follows:

- Gathers data from an endpoint application and puts it into the form the SFM expects.
- Uses a routing function (DFC) to send the data to the SFM.
- Checks the return value from the routing function:
  - If greater than zero (0), identifies the transaction condition.
  - If equal to -1, checks the clGetDfcErrno() value for a cause.
  - If -1 or less, performs the appropriate error alert or correction procedures.

Note For more information about routing functions, see the *e-Biz Impact ODL Guide*.

Setting up data formats

After the acquisition AIM receives all of the necessary data from the endpoint application, that data must be put into the format that the SFM expects from an incoming transaction. The SFM receives the transaction via one of several routing functions, which can send the data in a string or blob object. String and blob objects are ODL data classes.

1 Build data objects for the arguments used by the routing function. Data objects are data variables defined using MSG-IDE or TRAN-IDE:

- int *flv*

- string *src_ref*
- string *tranid* – used only with route_vrec().
- string *prodname* – used only with route_vprod().
- string *databuf* or blob *databuf*

2 Populate the data objects you built in the previous step. You can populate these data objects either before or in the same custom function from which you call the routing function.

If the value for any of the data objects is the same for every transaction sent by the acquisition AIM, then you can populate the data objects once during the program's initialization function, clinit(). Otherwise, populate the data objects with the correct values before each routing function call made by the program to the SFM. For example:

```
int route_vrec(int [flavor] flavor, string [in] src,
string [in] tranid, blob [in] data);

int route_vprod(int [flavor] flavor, string [in]
src, string [in] prodname, blob [in] data);

int route_veng(int [flavor] flavor, string [in] src,
string [in] engname, blob [in] data);

int route_recx(int [flavor] flavor, string [in] src,
string [in] tranid, blob [in] data, int [in] opts,
char [in] priority, string [in] fkey, string [out]
*serial, string [out] *errs, string [out] *status,
string [out] *dests, string [out] *sfmName, int
[out] *stmFlavor, string [out] *hostname);
```

Example

This example populates data objects for the route_vrec() routing function:

```
int rv;
int flav;
string src_ref;
string tranid;
blob dataBuf;

flav = 2;
src_ref = "acqaim1";
tranid = "tran1";
dataBuf = "this is my data";

rv = route_vrec(flv, src_ref, tranid, dataBuf);
```

where:

- *flv* – is the flavor of the SFM that should receive the data.

- *src_ref* – is a reference name for the acquisition AIM.
- *tranid* – is the transaction ID reference.
- *databuf* – is the data to send. In this example, name, address, and number are the defined data objects.

Return values

- The main controller that contains the SFM and ODL applications always returns a negative value to your acquisition AIM if it encounters a problem of some kind. If the return value is -1, it calls the clGetDfcErrno() function.
- If the e-Biz Impact cluster does not find an error, it returns whatever value the SFM sets as the return value.
- After the program issues a DFC command, to send or receive data, it should:
  a Check the return value for a negative value. If the return value is not a negative value, check for your programmed return values, if any.
  b If the return value is a negative value, check for one of the following conditions. Check for the integer value or for the mnemonic that is associated with the integer as listed in Table 2-2.

Table 2-1: DFC return values

Mnemonic	#	Description
RESERVED_FOR_DFCERR	-1	The cluster could not pass the transaction on to the SFM. Your program should call clGetDfcErrno() to troubleshoot the problem.
UNKNOWN_TRANCODE	-2	The SFM could not find the production name (route_vprod()) or the transaction ID name (route_vrec()) in its configuration file.
LOGFILE_FULL	-3	The SFM could not process the transaction because the log file was full.
TRAN_REFUSED	-5	The SFM refused the record because it is in refuse mode.
NOT_QUALIFIED	-6	The transaction data did not pass the field object validation or production object qualification checks in the referenced production object.
NO_DESTS	-7	There are no valid destinations for the production objects for which the transaction qualified. This occurs when the production objects uses the setDestName and/or the setDestNameData dynamic routing built-in filter functions to override the production object's configured destinations but the built-in filter function did not specify a valid destination.
INVALID_DEST	-8	The SFM could not process the transaction because an invalid destination was specified with a dynamic routing built-in filter function.

Error handling

The errors list in Table 2-3 can be returned by DFCs:

Table 2-2: DFC errors

Error code	Description
4300	General DFC error.
4311	The DFC timed out while trying to acquire and instance of the DFC server.
4312	The DFC timed out while waiting for the server to pick up the message (message removed from server).
4313	The DFC timed out while the server was processing the message (message still in server).
4321	The DFC function/flavor combination was not found in the route map.
4322	The DFC function/flavor combination was unavailable due to the availability configuration.
4331	The DFC connection to the child controller could not be established.
4332	The DFC connection to the peer controller, remote cluster, or SFM could not be established.
4340	General DFC error in the server.
4350	General DFC error in the client bridge.

Building ODL delivery AIMs

ODL delivery AIMs must define the following functions:

- servayt() – provides a response to the SFM when it checks to see if the AIM is active, and identifies the transaction last processed using the *lastID* file.

  For example:

```
int servayt(int [in, flavor] flavor, string [out] *pSerial, int [in]
     sfmFlavor)
```

- servproc() – used by the SFM to send a transaction to the delivery AIM.

  For example:

```
int servproc(int [in, flavor] flavor, string [in, out] *pSerial, string
     [in] src, blob [in] dataBuf, string [in] tranid, string [out]
     *errTxt, string [in] fkey, int [in] opts, char [in] priority, int
     [in] sfmFlavor)
```

In addition, an ODL delivery AIM must provide the return values listed in Table 2-1.

Table 2-3: Delivery AIM function return values

Return Value	Description
-1	Do not use a return value of -1, which is reserved by e-Biz Impact.
1+	A return value of 1 or greater indicates a successful DFC service function.
0, less than 0 (except for the reserved value of -1)	A return value of 0 or less than 0 indicates a DFC service function failure. The transaction for that destination remains in a PENDING status and the SFM goes into retry mode to dispatch the transaction again.
-998	Used by servproc() to tell the SFM to skip processing of the current transaction for a particular destination. The transaction is marked as SKIPPED for the destination and is moved to the completed log after the entire transaction is processed.
-999	Used by servproc() to tell the SFM to skip processing of the current transaction for a particular destination. The transaction is marked as CANCELLED for the destination and the entire transaction remains in the PENDING log.

You may want to have a delivery AIM cancel or skip a transaction if it finds an error that transaction production could not catch, such as assigning a duplicate part number. The transaction cannot be processed due to the error, but the delivery AIM that caught the error can continue to receive and process other transactions.

When a delivery AIM cancels or skips a transaction, the SFM marks the transaction for that destination as CANCELLED or SKIPPED. You can repair the suspended transaction using the Global Console. See *Monitoring e-Biz Impact* for detailed information about monitoring and log file maintenance.

CHAPTER 3

Using MSG-IDE

This chapter explains how to build ODL AIMs using MSG-IDE.

Starting MSG-IDE

To start MSG-IDE, select Start | Programs | Sybase | e-Biz Impact 5.4 | Msg-IDE. When the tool executes, the Message Interface Development and Browse windows open.

Building ODL AIMs

ODL AIMs contain both message interaction and supporting objects. Objects are listed in the left pane of the MSG-IDE Browse window.

ODL AIMs do not need to use all message interaction objects listed and, at a minimum, can contain only data and function objects. For example, an AIM that contains all objects, except communications objects, might receive data only through routing commands, but can, from within a function object, call the protocol object process() method and send data through the protocol object to the message frame objects. A message AIM that uses all message interaction objects can receive data through a communication object or routing commands.

MSG-IDE includes several primary and supporting object types. See “MSG-IDE objects” on page 4 for a description of each object type.

Note For more information about objects with user callable methods, see the *e-Biz Impact ODL Guide*.

Domains and files

Domains and files are found in the objects list, and you can add, delete, or rename a file or domain just as you would other objects.

Domains and files do not share all the characteristics of a standard object. Specifically, domains are logical containers that contain other objects, such as files that are added as to a specified domain.

Protocol objects

Protocol objects regulate the flow of data from the communication object to the message frame objects. It is the main container object for other message interaction objects, except data objects, in a message AIM. All data sent to a message AIM by the communication object enters the protocol object, which then controls the passage of data to the message frame objects.

The protocol object identifies the communications object, available message frame objects, control flow objects to run after the communication object is open and after it closes down, and the control flow object to run when the protocol object receives data from the communication object.

The protocol object executes the open control flow object after the communication object connects to the port, executing the close control flow object if the communication object loses the port connection, accepting data from the communication object, executing the preview control flow object after receiving data from the communication object, offering the data to the message frame objects for bidding, and submitting the data to the proper message frame object.

❖ **Defining a protocol object**

1 Select Protocol from the list of objects in the main MSG-IDE window and click Add. The Protocol Definition window appears.

2 Click in the Name field and type a protocol object name; for example, proto1.

3 In the Communications section, select the communications object used by the protocol object. To build a new communications object, click the ellipsis button. The Communications dialog box appears.

4 Complete these options:

Open: Select an existing control flow object from the list or click Detail to open the Control Flow dialog box and build a new control flow object. This is the control flow object run by the protocol object after Communication shows that it is up

After you build the control flow object, the name of its associated function appears next to the Run label.

5 Close: Select from the drop-down list or click Detail to open the control flow dialog box and build a new control flow object. This is the control flow object run by the protocol object when the communication object loses the connection or closes down before the application tells it to.

After you build the control flow object, the name of its associated function appears next to the Run label. If you want the communication object to re-establish the connection to the endpoint application, use restart() within the close control flow.

6 Preview Data: Select an existing control flow object from the list or click Detail to open the Control Flow dialog box and build a new Control Flow object. This is the control flow object run by the protocol object each time it receives data from the communication object and before it submits that data packet to the message frame objects for bidding. Use this control flow object to view or modify the data packet.

After you build the control flow object, the name of its associated function appears next to the Run label.

7 Choose one of the following:

- To view or modify all data that the protocol object has accumulated to send to the message frame objects for bidding, check the Accumulate Data check box.
- To view or modify the data packet that the protocol object just received from the communication object, deselect the Accumulate Data check box.

8 Choose one of the following:

The AIM Type identifies how the protocol object should handle incoming data. These options do not indicate the data format, only the way the protocol object handles the data.

- To accumulate the data received from the communication object, click Stream.

  If a message frame object does not make a successful bid on the data, the protocol object leaves the data in the blob, appends the next piece of data from the communication object onto the blob, and submits the accumulated data to the message frame objects. The protocol object does not clear its blob until a message frame object makes a successful bid on some part of the accumulated data.

- To have the protocol object not accumulate data, click Block.

  If a message frame object does not make a successful bid on the data, the protocol object clears its blob, places the next piece of data from the communication object into its blob, and submits the newly received data to the message frame objects.

9 At the bottom of the Frames pane, click Include to add an existing message frame object to the protocol object.

  The Frames pane displays all message frame objects available to the protocol object. The protocol object submits the data for bidding to the message frame objects based upon their order in the list.

  The Include Frame dialog box appears.

10 Select the message frame object to include from the drop-down list and click OK to return to the Protocol Definition window.

11 To add to protocol object to the project, click OK.

12 If you are editing an existing protocol object, click OK to update the object.

13 Use the start() method to add logic to the clinit() function to start the Protocol object (use `start() method...`).

  For more information, see the *e-Biz Impact ODL Guide*.

Defining control flow objects

Control flow objects define the actions to take and the functions to perform after a message frame object matches data. Control flow objects can be contained within a control flow object or separate within an AIM. The object generates a single function that executes all of the commands (the functions or methods) in the control flow. The commands in a control flow object act on the data itself or on other objects in the program that you want to manipulate based upon the data content. You must always pass the argument “blob *pb” to a control flow object in the Flow Arg. SLE of the control flow dialog box because the control flow object always receives the data in a blob.

Control flow object also provides a graphical representation of the commands, showing the order in which they are executed. A control flow object can be called by three objects. It can be called by:

- A protocol object after the protocol object receives a data packet from the communication object.
- A message from object after that message frame matches on data.
- Another control flow object

❖ Defining a control flow object

1 Select Control Flow from the list of objects in the main MSG-IDE window and click Add. The Control Flow window appears.

2 Click in the Name field and type the name you want to assign to this control flow.

This name, prepended by an underscore, becomes the reference name of the function generated by the control flow. For example, if you name the control flow cf1, you call it with the reference name _cf1. Use this reference name to activate the control flow object commands if you do not enter a name in the Run box.

After you build the control flow, this reference name appears in the Objects List in the Browse dialog box when the Function class is selected.

3 (Optional) To call the control flow object with an alternate name, type the name in the Run box.

For example, if you name the control flow cf1, but want to call it with the name myCF1, enter myCFI in this field.

When you use this field, this is the name that appears in the Objects list in the Browse dialog box when you select the Function class, and this is the name that appears in any drop-down lists from which you select function objects.

4 The Flow Argument Declaration defines the arguments that must be passed to the control flow object when it is called. These arguments depend on what type of object is calling the control flow. The following table describes the argument to use for each of the objects that can call a control flow object.

Object calling the control flow	Flow Arg. Declaration
Protocol: for an Open control flow	Leave this entry blank.
Protocol: for a Close control flow	Leave this entry blank.
Protocol: for a Preview control flow	blob *pb
Message frame object	blob *pb
another control flow object	user defined
Impact (to service a DFC command)	blob *pb

5 To put arguments in the Flow Arg. declaration field, select the appropriate argument from the Arguments menu bar option:

Argument	Description
Custom	Select this only if the control flow is executed by another control flow and is passed an argument other than blob *pb. Enter the required argument declarations in the Flow Arg. field.
Callback	Not applicable to a message AIM.
Validation	Not applicable to a message AIM.
blob *pb	Places blob *pb in the Flow Arg. declaration field. Select for all control flows, except Open and Close and for the exception stated under the Custom description above

6 Do one of the following:

- To make the control flow available only to objects in the same module, click the Exclusive to Domain check box.
- To make the control flow available to all objects in the entire project, deselect the Exclusive to Domain check box.

7 Use the following command specification fields to identify or define the functions for this control flow to execute and the objects for it to manipulate:

Field	Description
Description	For each command, an icon appears in the Commands list of the Control Flow. The Description field value appears next to this icon.
Statement	Click Statement when your function or object performs some action that does not require comparison, such as a function that builds a list or that gathers data from some external source. When you use a Statement, the control flow does not check the return value from the function or object method performed. It continues processing its remaining commands regardless of the return value for the function or method. Use Test if the control flow should stop processing or perform a different set of actions if the function or method returns a value other than a 1.
Test	Click Test when your function contains an "if" statement or when the control flow should perform one set of commands when the function or object method returns a value of 1 or another set of commands if it does not. For example, if the function "check that GL system is up" returns a value of 1, the control flow executes the "send data to GL" and "send ACK to client" statements. Otherwise, the control flow executes the "notify administrator that GL system is down" and "send NAK to client" statements.
Test/Loop	Click Test/Loop when your function contains a "while" statement. If the statement returns a value of 1, the control flow executes the commands in the path indicated by the 1 in the icon.
Stop	Click Stop to generate a simple return statement. Place it at any point at which your control flow should stop processing.

8 Select Function to build a command that uses a function or another control flow. If the function or control flow exists in the project, select it from the drop-down list.

If you entered a value in the Run field, that value is what appears in this list, not the control flow object reference name

To build a new function from within the control flow dialog box, choose from the following:

- Enter its name in the Function field, enter the arguments that should be used to call the function in the Command Arguments field, and then type the Function logic in the viewing area at the bottom of the Control Flow dialog box.
- Click Detail. From the Edit Function dialog box, define the function and then select File | Update to build the function and return to the Control Flow dialog box.

9 If the Function was selected, declare the datatype of each argument in the Command Arguments field in the Command Argument Declaration field.

For example, if the Command Arguments data object is named “my_int” (integer datatype), then this field has an argument declaration of “int my_int”.

Separate each argument with a comma. For example: `string name,string address,string city,int zip`

10 To execute a method on a particular object, click Object and select the object name in the Object field. Select the associated object method from the Method list, and enter the method's arguments in the Command Arguments field.

11 If the Object was selected, list the data objects that contain the information to be sent to the function or method in the Command Arguments field. In effect, you should enter the arguments as you would when making a call to a function or method. Separate each argument with a comma. For example: `name,address,city,zip`

12 The control flow logic displays in Commands pane in flow chart format as you build the control flow object. The control flow executes the functions and methods in the order in which they are listed in this box.

To modify or move commands in the Commands pane, do the following:

- Click the buttons below the Commands box to modify the display.

  These buttons let you add, update, or remove items from the display and move existing items to different locations in the control flow. This updates only the command display in the control flow, not the associated function.

  Click Update if you modify a command in the Commands box. If you do not, a warning message appears.

- Click the arrow buttons to move a command from one location to another in the control flow. Highlight a command in the Commands box and click the arrow pointing in the appropriate direction. The object moves one position with each click

13 Select File | Close to close the control flow without updating.

Defining message frame objects

Message frame objects define the structure of the data from the communication object that the program should act upon and the control flow objects to execute once that data arrives.

A message frame object defines the format of a specific piece of the incoming data message, describing the byte order sequence of the data to match. A message AIM can use as many message frames as necessary to identify the structure of the incoming message (data). The message frame object identifies the start of the data, datatype/length, data end, and the control flow object to execute when incoming data matches the message frame definition.

❖ **Defining a message frame**

1 Select the Message Frame object and click Add. The Frame Matching window appears.

2 Click in the Frame Name field and enter a name for the message frame object that is descriptive of the type of message being framed by this object.

3 To define the Message Frame, click the appropriate button and complete additional information in its associated field.

The button options are described in the following table:

Button	Description
Any	Select this option to match on any type of data for the specified number of bytes. Enter the number of bytes in the entry field.

Button	Description

All Data Until	Select this option to match any type and amount of data until it reaches whatever value is specified in the next line of the frame definition. To stop matching when the frame finds a set of values instead of a single value, define a definition for each value, in the order you want them matched, after the "All Data Until" statement. Then put an "...end All Data Until" statement after the definition for the final value.
	For example, if you want the Frame to stop matching on data when it finds a CRLF combination, but not when it finds just a CR or just an LF, place the following definitions in the message frame object:
	`All Data Until..`
	`Control Character CR`
	`Control Character LF`
	`..end All Data Until`

end All Data Until	Select this option to stop matching upon encountering any type of data.
Hex value	Select this option to match on the hex value specified in the entry field.
Decimal value	Select this option to match on the decimal value specified in the entry field
Octal value	Select this option to match on the octal value specified in the entry field.
ASCII String	Select this option to match on the ascii string specified in the entry field.
Control Character	Select this option to match on the control character specified in the entry field.
Var String	Select this option to match on the value of the selected char, string, or blob variable. The drop-down list contains all char, string, and blob variables declared in the Define dialog box of the project data object. **Note** The values of the Var String and Var length variables must be initialized in clinit(). You can modify the variables by using the message frame object reset() method within another function. See the *e-Biz Impact ODL Guide* for more detailed information about the reset() method.
Var Length	Select this option to match of the value of the selected short, int, or long variable. Use this option to make a variable out of the number of bytes delimiting the message. Use the Any option to hard code the number of bytes. The drop-down list contains all short, int, and long variables declared in the Define dialog box of the project data object

4 Click Add to add the definition to the message frame object.

5 Click the Non-recursive scan check box if you do not want the message frame object to perform a recursive scan on the data.

The message frame object performs recursive scanning by default in case some interference, such as line noise, causes the communication object to receive only part of a data packet and then the entire packet if it is re-sent.

In a recursive scan, after the message frame object matches on part of the data, it checks within the matched area to see if it can match on a smaller part of that data. The Message Frame continues to check within the matched data for another match until it finds the smallest possible part of the data to match.

Example: The Protocol Object submits this data to the Message Frame Object for bidding. The Message Frame Object is looking for an STX and then all data until an ETX.

```
STXthisSTXthis is aSTXthis is a test.ETX
```

In a recursive scan, the Message Frame Object first matches on the entire piece of data, “STXthisSTXthis is aSTXthis is a test.ETX”. It then looks for another match within that piece of data and finds “STXthis is aSTXthis is a test.ETX”. It then looks within that piece of matched data for another match and finds “STXthis is a test.ETX”. It then looks for another match within that piece and does not find one, so it bids on the final piece of the data that it matched on. In a non-recursive scan, the Message Frame Object matches on the entire piece of data, “STXthisSTXthis is aSTXthis is a test.ETX”, does not look any further within the data for another match, and bids on the entire piece of data.

6 The Frame Definition box displays the frame definition, which describes what the data looks like that the message frame object should match on. Each part of the definition defined in the Options box displays in this box.

- Click Add to add the currently defined option to the message frame definition.
- Click Remove to remove a selected line from the message frame definition
- Use the spin arrows to change the order of lines in the Frame Definition box.

7 Click the Control Flow down arrow and select an existing control flow object from the list.

The control flow object that the message frame object should run after the message frame makes a successful bid on the data.

8 Click Detail to open the control flow dialog box and build a new control flow object.

The name of the function associated with the control flow appears next to the Run label.

9 When you finish completing the fields or updating the message frame, click Close to close the Frame Matching dialog box.

The message frame object is added to the project or the message frame is updated with the modifications.

Defining communication objects

The message AIM uses communication objects to handle communications with the endpoint application. Once the message aim is initialized, the protocol object launches the communication object to establish the endpoint connection. The communication object then receives data from the endpoint and passes it to the protocol object to be processed using defined frame objects. The data is send to the endpoint programmatically using the communication object's send() method.

To set up the communication mode, use the Configurator. For more detailed information, see the *e-Biz Impact Configurator Guide*.

❖ **Defining a communications object**

1 Select Communications from the list and click Add. The Communications window appears:

2 To configure the communications object, enter a name for the object in the Name field.

When you configure the ODL Application for the AIM in the Configurator, use the Icon Name as the name of the ODL Resource.

3 Specify the Icon Name for the communication object.

Note The Icon Name is required for both UNIX and Windows operating systems.

4 Click OK to add the new communications object to the project.

Note See the e-Biz Impact ODL Guide for details about the kill(), restart(), and send() methods, which are associated with the communications object.

Defining data objects

Data objects provide data variable definitions and contain data that you may want to manipulate or access at a later point in the program. A data object is any data variable defined with MSG-IDE or TRAN-IDE.

Use the Define dialog box to build data objects to:

- Build a DFC command for gathering or sending data.
- Add a symbolic name definition.
- Build your own structures or class definitions.

As you build a definition, you can see what the program generates in the viewing area at the bottom of the dialog box. Expand the size of the viewing area, if necessary, to view an entire definition

❖ **Defining data objects**

1 To access the Define dialog box, select Data from the Classes list and click Add.

2 For the Basic Datatype, complete the following fields:

Type	Description
char	Character: a single character.
short	Short Integer: a whole number in the range of -32767 to +32767
int	Integer: a whole number in the range of --2147483647 to +2147483647
long	Long Integer: a whole number in the range of -2147483647 to +2147483647
float	Floating Point: a floating point number.
string	A null-terminated character string.
cptr	A real C memory pointer.
iptr	For internal use only.
void	A void pointer
decimal	A decimal number, like 123.45
blob	A binary large object block. A collection of bytes that can include null characters, and is terminated by length.
clNdo	Use in conjunction with tree data, supported by the clNdo object.
clot	Use in conjunction with transports, supported by the clot object.

Type	Description
clmap	Store string pairs in a <key, element> format.

3 From the Class drop-down list, select to build an object for the selected class.

Only the classes that are useful in a message AIM are listed below. Except the string class, you can also build any of these objects directly from the Browse dialog box.

Class	Description
Timer	To build a timer object.
Control Flow	Builds a control flow object.
Message Frame	Builds a message frame object.
Protocol	Builds a protocol object.
Communications	Builds a communications object.
string	Builds a string object.
Blob	Builds a blob object
Production	Use TRAN-IDE to edit Production Objects. Not used by MSG-IDE
clNdo	Use to build an NDO object.
clot	Use to build a transport object.
clmap	Store string pairs in a <key, element> format.

4 Click Distributed Function to define a Distributed Function Call that this program makes to an AIM or an instance of SFM.

These are blocking function calls; that is, after the program issues this DFC command, it waits for a return response from the program that services the function call before it continues its own processing.

To build a DFC command, type its name in the Name field and click either OK or Accept. The Distributed Function Declaration dialog box opens, where you define the DFC command arguments.

5 Click Symbolic Name to add a local or global Symbolic Name.

A symbolic name is a means for substituting a name for a particular value, usually a number. Symbolic names allow you to reference a number when the value of the number may change. For example, if you set up a symbolic name for the maximum number of cattle you could keep in a pen (#define MAXINPEN 10), and used the symbolic name (MAXINPEN) instead of the number (10) throughout your program, then, if the pen size changes so you can only put 5 cattle in a pen, you only have to change the symbolic name value.

You can use a symbolic name Data Object in Custom, Callback, and Validation functions. If you set Exclusive to Domain ON, the program builds a #lcldefine statement that is available only to the objects and functions in the current module.

If you have Exclusive to Domain set OFF, the program builds a icldefine statement that is available to all the objects and functions in the entire project. Only numeric values are supported.

6 Click Custom to define data objects that are not available with the basic datatype option. Use the area at the bottom of the Define dialog box to enter your code.

7 Type the user-assigned name for this definition

In all cases, make this name unique within the domain. If you have Exclusive to Domain deselected for building a global definition, you should make the name unique to the project. Names must be a maximum of 32 characters, contain only A-Z, a-z, 0-9, and underscore characters, and begin with an alpha character.

8 In the Initial Value field, type the initial value for the data object.

This value is useful only when defining a basic datatype or a symbolic name.

9 In the Array Size field, type the size of an array.

This field is only available for definitions of basic datatypes.

10 Do one of the following:

- (Default) Select the Exclusive to Domain check box to allow the data object to be accessed only by other objects and functions within the same domain.
- Deselect the Exclusive to Domain check box to allow all objects and functions in the project to reference the data object.

11 The viewing area at the bottom of the Define dialog box displays the Data Object you are currently creating and, in some cases, allows you to edit that Data Object definition.

If the definition is longer than two lines, either use the scrolling arrows or enlarge the viewing area. To enlarge the viewing area, move the cursor over the bottom of the dialog box until it displays as a two-way arrow. Then hold the left mouse button down, drag the cursor to the size of the dialog box you want, and release the mouse button.

12 Do one of the following:

- Click OK to build the data object and return to the Browse dialog box.
- Click Accept to build the data object and remain in the define dialog box.
- Click Cancel to return to the Browse dialog box without building the data object.

Defining blob objects

A blob (Binary Large Object Block) is a collection of bytes containing arbitrary values that may include null characters. It is terminated by a length value. To define a Blob Object, use the format:

```
blob obj_name;
```

Generally, a Blob object maximum size is the same as the maximum size of an integer. This size is dependent on the computer architecture, whether 16 bit, 32 bit, or 64 bit. For most computer systems, the maximum size is 4 GB.

There are two ways to build a blob object:

- Highlight “Blob” in the Classes list of the Browse dialog box and click Add.
- When the Define Window opens, select Class, then select Blob from the associated drop-down list

Note See the *e-Biz Impact ODL Guide* for details on blob methods.

Defining distributed function objects

A Distributed Function Object contains the definition of a Distributed Function Call (DFC command) that the AIM makes to another application within the Impact environment. In the DFC command, you define arguments for the information going out to the Impact server and the receiving application and for the information returning from it. The order of the arguments in this window must exactly match the order of the arguments in the Impact receiving DFC function. As you build the arguments, they appear in the idempotent statement displayed in the viewing area at the bottom of the window. When you delete an argument from the Argument List, it is also removed from this statement. Enlarge the bottom of the window to view the entire statement if it exceeds a few lines.

❖ **Defining a distributed function object**

1 In the Distributed Function Declaration window, click in the Distributed Function field and type the distributed function call (DFC) command.

If you accessed the Distributed Function declaration box through the Define window, this field is disabled and displays the last entry to the Name drop-down box in the Define window.

After you build a DFC command, if you select Accept, focus returns to this drop-down box, and you can enter the name of another DFC command.

2 Choose one of the following:

- Select the Exclusive to Domain check box to make the DFC command exclusive to the domain, or static; that is, only objects and functions in the same domain can reference the DFC command.
- Deselect the Exclusive to Domain check box to make the DFC command considered global to the project, meaning that objects and functions in other domains in the same project can reference it

3 Select the Function Wrapper check box to generate a function that issues the DFC command, checks the return value from the function, and displays an error message if it detects a problem.

The function name is identical to the DFC command name, but is prepended with an underscore. The Argument Name field becomes a drop-down list from which you can select predefined data objects as arguments.

4 Click in the Argument Name field and type the user-assigned name for this argument in the Name field.

This name does not need to match the name of the argument used in the destination AIM.

5 Select one or more of the following Attributes:

Attribute	Description
In	The sending AIM fills this argument with a value before issuing the DFC command.
Out	The receiving application fills this argument. The destination AIM fills this argument upon receiving the DFC command and before the argument set returns to the sending AIM.
Flavor	Impact uses the flavor in the idempotent statement to choose a specific function from its tables to service the DFC command. Only one argument per DFC command can have a Flavor attribute.

Each argument must have one or more of the attributes check boxes selected.

6 Select the Array check box and use the associated field to define an array.

7 Select the Pointer check box if this DFC argument is a pointer.

8 In the Argument List box, which displays a list of the arguments built for this DFC command, do any of the following:

- To delete an argument from the list, highlight the entry and click Delete.
- To modify the presentation sequence of the arguments, use the spinner buttons.
- To update changes to the argument list, click Update.

9 Click Add Arg when you complete your definition of an argument to add it to the arguments list.

You can use a maximum of 100 arguments per DFC command.

Viewing Area

The viewing area is at the bottom of the Distributed Function Declaration dialog box. It displays the DFC command in its current state.

- If the DFC command exceeds two lines, use the scrolling arrows to display the rest of the definition, or you may enlarge the viewing area.

- To enlarge the viewing area, move the cursor over the bottom of the dialog box until it displays as a two-way arrow. Then hold the left mouse button down, drag the cursor to the size of the dialog box you want, and release the mouse button.

Note See the *e-Biz Impact ODL Guide* for more information on SFM and DFC interfaces.

❖ **Using the Viewing Area**

1 Click Accept to add the distributed function object to the current file and remain in the Distributed Function Declaration box to build another object.

If you build the object from the Define dialog box and click Accept in the Define dialog box, you return to that dialog box.

2 Click OK to add the distributed function object to the current file and return to the Browse dialog box.

Defining database interface objects

Database interface objects allow ODL programmers to access external databases. You can use a database interface object to:

- gather data from a database and store the data in data objects
- Gather data from data objects and store the data in a database
- Transform data in a filter. Field objects can be compared with database entries and modified if necessary.
- Check data in a qualifying object. If the database interface object returns any rows, the qualification is passed.

You can write SQL to use the contents of a data object as a comparison point for selecting rows of data, then bind columns of the result row to data objects and manipulate data objects with the callback function. For example, you can put a last name in a data object, then use the last name to draw a row from a database of last name-salary-phone number information. Bind the salary column to another data object, and use the callback function to send the last name and salary to the message broker for distribution to an endpoint application.

❖ **Defining a database object**

1 To view the database interface, select Tools | Database Interface from the MSG-IDE toolbar.

2 Select Database Interface from the Classes list and click Add.

3 Click in the Obj Name field and type a name for the database interface object.

4 Click in the Dsn field and type the DSN name setup on the current system.

5 Click in the User field and type the user name to access the database defined in the DSN.

6 Click in the Password field and type the associated password.

The *nnsyreg.dat* file entry should look similar to the following example, for a SQL 8.0 database on Windows 2000:

```
OTContext.InputContext
NNOT_CTX_TMID= OTTestTransactionManager
NNOT_CTX_ENFORCE_TX= TRUE

OTContext.OutputContext1
NNOT_CTX_TMID= OTTestTransactionManager2
NNOT_CTX_ENFORCE_TX= TRUE

OTContext.OutputContext2
NNOT_CTX_TMID= OTTestTransactionManager
NNOT_CTX_ENFORCE_TX= TRUE

Transport.TransportInput
NNOT_SHARED_LIBRARY= dbt26mqs
NNOT_FACTORY_FUNCTION= NNMQSQueueFactory
NNOT_TIL_OPEN_SESSION_ID= MQSession

Transport.TransportError
NNOT_SHARED_LIBRARY= dbt26mqs
NNOT_FACTORY_FUNCTION= NNMQSQueueFactory
NNOT_TIL_OPEN_SESSION_ID= MQSession2

Transport.TransportOutput1
NNOT_SHARED_LIBRARY= dbt26mqs
NNOT_FACTORY_FUNCTION= NNMQSQueueFactory
NNOT_TIL_OPEN_SESSION_ID= MQSession

Transport.TransportOutput2
NNOT_SHARED_LIBRARY= dbt26mqs
```

```
NNOT_FACTORY_FUNCTION= NNMQSQueueFactory
NNOT_TIL_OPEN_SESSION_ID= MQSession

TransactionManager.OTTestTransactionManager
NNOT_SHARED_LIBRARY= oti26mqstm
NNOT_FACTORY_FUNCTION= NNOTMQSeriesTXManagerFactory
NN_TM_MQS_QMGR= QUEUEMGR

TransactionManager.OTTestTransactionManager2
NNOT_SHARED_LIBRARY= oti26mqstm
NNOT_FACTORY_FUNCTION= NNOTMQSeriesTXManagerFactory
NN_TM_MQS_QMGR= QUEUEMGR

Session.MQSession
NNOT_SHARED_LIBRARY= dbt26mqs
NNOT_FACTORY_FUNCTION= NNMQSSessionFactory
NNMQS_SES_OPEN_QMGR= QUEUEMG

Session.MQSession2
NNOT_SHARED_LIBRARY= dbt26mqs
NNOT_FACTORY_FUNCTION= NNMQSSessionFactory
NNMQS_SES_OPEN_QMGR= QUEUEMGR
```

7 The Statements pane displays a list of the statement objects in the database interface object. A statement object comprises SQL statements and other settings that perform an action involving the database. The statement objects apply only to the database to which the database interface object is linked.

- To create a new statement object, click New.
- To view an existing Statement Object, select the statement and click Details.

8 Click the drop-down list in the Isolation Level field to select an isolation level.

The isolation level is the method used to deal with concurrent data calls to the database. Refer to the ODBC manuals provided by your DBMS or database middleware vendor for information on isolation levels.

9 To select Status Scope, click its check box.

10 Click OK to accept the settings in the database interface object dialog box and close the dialog box.

Using a SQL template and a datalink

`ClDbStmt` is the ODL representation of a database-specific SQL statement. A SQL statement is constructed with the creation of a `clDbStmt`, which is done using MSG-IDE or TRAN-IDE.

❖ Building a SQL statement

1. Start MSG-IDE. Select Start | Programs | Sybase | e-biz Impact 5.4 | Msg-IDE.
2. If Database Interface does not appear in the object list on the left of the MSG-IDE Browse window, select Tools | Database Interface in the Message Interface Development window.
3. In the Browse window, select Database Interface in the object list and click Add. The Database Interface Object window opens.
4. Click New. The SQL Statement Builder window appears.
5. In the Stmt Name field, type a descriptive name for the statement object that you are building.
6. In the Stmt ID# field, type the ID number of the statement object that you are building. The ID number allows other objects to reference the statement object.
7. In the Tables section, click the table that you want to use. The columns in the selected table appear in the Columns list. To insert a table name in the SQL syntax, double-click the table name.
8. Click a column in the list to display the column's datatype in the status bar at the bottom of the window. Double-click the column name to insert it into the SQL syntax.
9. Click a procedure in the list to display its parameters in the status bar. To insert a procedure into the SQL syntax, double-click the procedure. The procedures section contains a list of the available procedures in the database. A procedure is a piece of user-defined logic contained in its database.
10. To insert the contents of the data object into the SQL syntax, double-click the Data Object. The Data Object name enclosed by << >> appears in the Syntax Box. This is an output Object. Field objects, indicated by type clTrFld, are read-only.

11 To insert the return value from a procedure into the data object in the SQL syntax, press Shift and double-click the Data Object. The Data Object name enclosed by >> << appears in the Syntax Box. This is an input object. For example, >>object<<

12 To insert the content of the Data Object, while at the same time inserting the return value from a stored procedure into the Data Object, type the name of the stored procedure, enclosed with <> <>. For example, <>object<>

13 Insert SQL into the syntax box by clicking the SQL verbs and the keypad, and by double-clicking items in the Tables, Columns, Procedures, and Data Objects lists.

To replace text in the box, highlight the text you want to replace and click the replacement

The Database Interface Object statement can contain more than one SQL statement. The maximum size for the SQL in this box is 2000 bytes.

Buttons are provided for common SQL syntax and functions. Click these buttons to build the SQL syntax with a minimal amount of typing

Datalink types

The following table lists all Datalink types.

Datalink	Type	Description
<<odl_variable_name>>	Input	Datalinks of this type retrieve the value of the corresponding ODL variable at execution time and substitute the value where the datalink is placed in the SQL statement.
>>odl _variable_name<<	Output	Datalinks of this type are written with the data sent back from the database on execution of the statement. Use this datalink only with stored procedure SQL statements.
<>odl_variable_name<>	Input Output	The value from the ODL variable associated with the datalink is used at execution time. Data returned from the database is also written to do the ODL variable. Use this datalink only with stored procedure SQL statements.
<!odl_variable_name!>	Input Output	Data from the ODL variable is substituted/expanded in place of the datalink literally (as text characters) at execution time.

Note Outbound and in/out bound datalinks used in stored procedures are not populated with data from the system database until all rows from the result set (if any) are fetched. To ensure population of the datalink, set the statement to fetch via the multifetch option.

SQL statement options

You expand the Statement Object definition by changing the source for SQL, expanding the SQL syntax, or automatically committing to database changes.

❖ **Defining SQL statement options**

1 In the SQL Statement Options dialog box,

2 Click the down-arrow in the SQL Data Obj field to select an existing Data Object, or click Detail (...) to open the Define window and create a new Data Object

If your SQL commands vary depending on the result of other operations, use a Data Object to store the SQL and reference the Data Object. The SQL commands in the Data Object selected in this dialog box is executed. The SQL commands in the Syntax box of the SQL Statement Builder dialog box is ignored.

3 Click the auto commit check box to automatically commit the changes to the database upon completion.

This guarantees the changes are permanent without a manual confirmation.

4 Click the inline data obj. references check box to enable inline Data Object references.

References in the SQL Data Object can be expanded by substituting values for existing parameters.

For example, given {string foo;} and {foo="nextus"}, then "select * from sales where id like <<foo>>" is expanded to "select * from sales where id like 'nextus'".

This is useful if your ODBC or database does not support extended parameters.

5 Do one of the following:

- Click Done to accept the settings.
- Click Cancel to exit without accepting the settings.

SQL statement result options

In the SQL Statement Result Options dialog box, you can assign a name to the result set of the statement object, execute a function on the result set, define column and row separators, determine a maximum number of rows to be included in the result, and specify other aspects of the statement output.

❖ Defining SQL statement result options

1 Click the down arrow in the Data Obj field to select an existing Data Object or click Detail to build a new Data Object.

Individual column data can be bound to Data Objects using the Column Bindings dialog box.

The Data Object stores all of the result rows generated by the Statement Object. The result rows include all separators, and are concatenated as they are processed,; ensure that the Data Object is large enough to hold all of the result data. Use a Data Object type of either BLOB or string as both grow as needed.

2 Click the down arrow in the Callback field to select an existing function to execute against each result row, or click Detail to build a new function. The function executes after each result row is gathered and acts on the data in the new row. The following table lists possible callback events generated by the Database Interface Object.

Event	Condition
DBE_SELECT	Got a row of data on a select or stored procedure.
DBE_ERR_OOM	Out of memory error.
DBE_ERR_SQL	SQL syntax error at run time.
DBE_ERR_ARG	Invalid arguments.
DBE_ERR_NOROWS	No rows affected on insert/update/delete command.
DBE_NOTIFY	Insert/update/delete completed successfully.
DBE_EMPTY	SELECT has returned empty result set.
DBE_RESULT	SELECT has returned entire result set, which has at least one result row.

See “Defining function objects” on page 52 for more information about building a function.

3 In the Col. Separator field, type the string to use as a column separator. The string can include Escape sequences.

4 In the Row Separator field, type the string to use as a row separator. The string can include Escape sequences.

5 In the Maximum Rows field, type the maximum number of rows to be included in the result-set row count.

The default value is 0, which means all rows. The maximum value is 2,147,483,647.

6 Click the No trunc check box to leave trailing white space on each result column.

7 Click the Multirow check box to return all result rows. Otherwise, only the first result row is returned.

8 Select the No Fetch option if you do not want to automatically fetch each result row after executing the Callback function. Otherwise, result rows are automatically fetched.

You can use the fetch( ) method to step through the result set. The fetch() method triggers the callback function each time it is called.

9 Select the Skip Last Row Sep option if you do not want a row separator placed at the end of the last row.

10 Select the Skip Last Col Sep option if you do not want a column separator placed at the end of the last column in each row.

11 Do one of the following:

- Click OK to accept the settings.
- Click Cancel to exit without accepting the settings.

SQL bindings

In the SQL Bindings dialog box, when you include a column-generating procedure in the SQL syntax, you can move a piece of column data into a Data Object after each result row is returned. This Data Object can then be used by the SQL Statement Result Options callback function or other objects outside the database interface object.

❖ **Binding column data to a data object**

1 Click the column name in the Column Box, choose a data object from the Data Object drop-down list, and click Set.

The column box shows which column data is bound to which data object. The columns appear on the left side of the box and the Data Objects bound to those columns appear on the right side of the box. The link is represented by ==>.

2 Click on the down arrow in the Data Object field to open a drop-down list of Data Objects and select the Data Object that you want to bind to a column.

3 Click the Auto Bind check box to automatically bind any column data to a Data Object that has the same name as the column.

For example, if the Auto Bind box is selected, the result set has a column named FName, and you have a Data Object named FName, the contents of the column is placed into the Data Object of the same name every time a result row is returned. The name of the column must be identical to the name of the Data Object. Selecting Auto Bind automatically matches identically named column and Data Objects and disallows a match in which the names vary in any respect.

4 Do one of the following:

- Click Accept to accept the settings.
- Click Clear to clear all the bindings on all columns.
- Click Cancel to exit without accepting the settings.

SQL statement syntax

SQL statements are passed to the underlying system database when executed, therefore, they must conform to the system database syntax. SQL statements can contain, input, inline, and datalinks. Embedded datalinks within the SQL statement are translated or converted to appropriate system database values.

SQL statements, such as select, insert, update, and drop, should not contain outbound (output, in-out) datalinks. Use these with Stored Procedures only.

Example of a SQL statement with a datalink:

```
Select * from customers where id = <<odl_int>>
```

Stored procedure syntax

A stored procedure template syntax requires:

- Stored procedure name on the system database.
- The correct number of parameters the stored procedure on the system database expects.
- An optional output datalink to store the return value of the procedure.

Stored procedure templates can contain all four types of datalinks.

The format for the stored procedure template is:

```
{ >>rv<< = call storedProcedureName (parameters) }
```

where `rv` is an optional datalink to hold the return value of the stored procedure. Not all stored procedures have a return value.

storedProcedureName is the name of the stored procedure on the system database.

parameters are parameters for the stored procedure. Parameters can include string, numeric literals, or any of the four datalink types.

Examples of a stored procedure call:

```
{ call sp1('foo', 'bar', 1, 2, 3, <<int_datalink>>) }
{ >>int_rv<< = call sp2(>>string_value<<,<int_value>>)}
```

Advanced datalink features

New datalinks features are transparent to existing and older project files. The new features are designed to give you more precise control of the data being sent to and retrieved from the system database. Use of the new features is optional.

A datalink consists of four parts:

- Datalink direction. One of four types: in, in-out, out, and inline.
- (Optional) Pre-cast to represent the datalink as a particular database column type. This is used for input type datalinks to represent ODL data as a different datatype in the system database.

  An example is a datalink that a string type in ODL. You can represent this as a type of varchar data to be used in a SQL statement instead of the default type of char. Casting is used in this situation.
- ODL variable name the datalink is bound to. Variables used as datalinks must be of global scope.
- (Optional) Post-cast to represent data from the system database as a particular ODL datatype. Use this for outbound datalinks to represent the data retrieved from system database as a particular ODL datatype.

  An example is in a stored procedure, to store binary values from an outbound datalink as sequence of null terminated characters to store in an ODL string, cast this into a char.

All datalinks have a default ODL type to system database type conversion. These are listed in the following table.

ODL datatype/SQL type	Description
blob	SQL_char
float	SQL_double
char	SQL_char

ODL datatype/SQL type	Description
unsigned char	SQL_tinyint
short	SQL_tinyint
unsigned short	SQL_tinyint
int	SQL_integer
unsigned int	SQL_integer
long	SQL_integer
unsigned long	SQL_integer
string	SQL char

ODL datatypes not listed in conversion table above are treated as SQL_char.

Pre-cast and post-cast syntax

Pre-cast

Datalink syntax:

```
<< (Pre-cast) datalink_name [Post-cast] >>
```

Pre-cast represents ODL datatypes in the datalink as a different datatype on the system or a datatype with a particular size and precision, which is also known as “sql_digit.”

Pre-cast is optional, it may not appear in the datalinks.

Syntax and options

Allowable syntax for pre-cast:

Note The enclosing parenthesis are required.

(*cast_type*)

(*cast_type*, sql_size)

(cast_type, sql_size, sql_digit)

Possible *cast_type* for pre-cast are:

- char
- varchar
- longvarchar
- binary
- varbinary
- longvarbinary

sql_size and sql_dIgit

The sql_size is the user-specified size for the data being sent to the system database. The sql_size can be used to limit the amount of data being sent to the system database if the specified size is smaller than the ODL data size.

sql_digit is used for precision.

For fixed length datatypes, these two options are disregarded.

An example of a pre-cast datalink is:

```
<<(varchar, 10)odl_string>>
```

Datalink Syntax:

```
>> (Pre-cast) datalink_name [Post-cast] <<
```

where:

`>>` is the datalink type. This could be one of four: <<. >>, <>, <|

`Pre-cast` is optional.

`datalink_ name` is the ODL variable name bound to the datalink.

`Post-cast` is optional.

Post-cast

Post-cast fine tunes the results retrieved from the system database so they can be stored in outbound datalinks in a certain way.

Note Use post-cast only with outbound or in-out datalinks. It cannot be used with inbound or inline type datalinks.

Syntax and options

Allowable syntax for post-cast:

Note The enclosing brackets are required.

[*cast_type*]

[*cast_type*, varible_buffer_size]

Possible *cast_type* for post-cast:

- char
- long
- binary
- float

The `variable_buffer_size` is the buffer size to retrieve from system database.

An example of a post-cast datalink is:

```
>>odl_string[char, 10]<<
```

Additional performance considerations are:

- Each clDbi object owns multiple clDbStmts during run time. At any given time, only one clDbStmt can be executed. ClDbStmts and associated SQL statements not in use are cached until they are selected to be the active statement and executed.
- Caching of statements significantly improves performance, however, ClDbStmts and associated SQL statements can only be cached in the following scenarios:
  - The SQL statement in the ClDbStmt does not contain inline datalinks.

    Because values of inline datalinks are expanded within the statement and values may change during run time, this type of SQL statement cannot be cached.
  - The SQL statement within the ClDbStmt is constructed and defined by the SQL Statement Builder before run time.

    ClDbStmts using an ODL datalink string to hold SQL statement cannot be cached because its value may change during run time.

Defining function objects

Within a control flow, a function object defines the actions to take, either on the data itself or because of the data presence. When not within a control flow, a function object defines the actions to take to service a DFC command. Function objects are user provided routines within which you may perform all kinds of data manipulation and actions in the Object Definition Language (ODL).

When you select Function in the MSG-IDE object list and click Add, the Edit Function window displays.

When you add a function, you select the type of function you want to create by clicking one of the buttons on the top of the window. The following table lists the available function types:

Function type	Description
Initialize	Specifies any actions the ODL application must perform when initialized by executing clinit(). Start your protocol object in this function.
Acquire	Executes the clacquire().
Deinitialize	Specifies any action the ODL application must perform when deinitialized. The program executes this function after calling clQuit().
DBCallback	Builds a function that manipulates the data received from each result row of a SQL statement object. The function executes after each result row has gathered events and data in the new row.
Custom	Specify the actions to take on application data or on other objects in the AIM that you want to manipulate based on data content. You can call custom functions from within a control flow object. If necessary, you can use the same custom function in multiple control flows.
Callback	Acts as a control object. You can assign the function name, but you cannot modify the return value or arguments received. Attach the function to a control object by assigning it to the Callback Function property for that control. You can have multiple callback functions in an application, but each must have a unique name.
Validation	Used to perform validation on the data beyond normal content checking. For example, a validation function could check for a range of numbers rather than just check for numeric data entered into a numeric field.

❖ **Defining function objects**

1 Select Function from the Classes list and click Add

The Edit Function window appears.

2 Select File | Update to check the syntax of the ODL in the function.

If MSG-IDE finds no errors, it adds the function to the current domain, or updates it if editing an existing function, and returns focus to the Browse dialog box. If it finds an error, it displays a Syntax Errors window that lists the line(s) that contain syntax errors, along with a description of the error. Click Done in the Syntax Errors window to return to the Edit Function dialog box and fix the detected error.

3 Select File | Exit to exit the Edit Function dialog box without checking syntax or adding the function.

Defining I/O file objects

An Input/Output (I/O) file object provides the ability to access disk files and to create, edit, and delete these files. The I/O file object references the actual file on the disk and defines access permissions for the file. After you create the file object within MSG-IDE, use methods to create or delete the file and to manipulate its contents.

I/O file objects also give you the ability to read the contents of a directory. In this case, the I/O file object references the directory rather than a specific file. You can then use openDir() and readDir() to create an internal list of the files that exist within the directory and to access each file in the list. See the *e-Biz Impact ODL Guide* for more information on the file object methods.

❖ **Defining an I/O file object**

1 Select Tools | IO File from the MSG-IDE menu bar. The I/O File Object window appears and I/O File now appears in the Classes list.

2 Select I/O File and click Add.

3 Click in the Object Name field and type a name for your I/O file object. The name cannot contain a period (.) or a space.

4 Click in the Associated File field and type the file name.

- When the I/O file object references a file, this is the name of the file to access or to create.
- When the I/O file object references a directory, this is the name of the directory to access.
- If the file or directory is not in the current working drive or directory of the ODL application, include either the full or relative path to the file.
  - Use a forward slash in the path for both Windows and UNIX.
  - This field is limited to 255 bytes for the actual filename and 1024 bytes for the path to the file.

5 Choose one of the following I/O Options:

- Read Only: The I/O file object can only read the file.
- Write Only: The I/O file object can only write to the file.
- Read and Write: The I/O file object can read and write to the file.

6 Select from the following Write Options:

Use these options to specify how the I/O file object writes to the file. These options are not available when Read Only is selected in the I/O Options box.

You can select only one option, or you can select Create with either the Append or Truncate option. In the latter case, the I/O file object creates the file only if it does not currently exist.

- Append: select to append new data to the end of an existing file.
- Truncate: select to delete the contents of an existing file before writing new data to the file.
- Create: select to create the file. Select the Exclusive check box to not create the file if a file of the same name already exists in the location specified in the Associated File field. When set and the file already exists, the I/O file object returns an FEEXIST error.

7 Select from the following Create Permissions options:

These options are used only when the I/O file object creates a file; they do not affect the permissions set on existing files.

- Owner: Select the choices in this column to specify permissions of the file owner. The file owner is the controller running the application.
- Group: Select the choices in this column to specify permissions for other users in the owner's group. The owner's group is the group of the controller running the application.
- Other: Select the choices in this column to specify the permissions for all other users.

8 Click the Binary I/O check box to open or transfer the I/O file object associated file in binary rather than text mode.

This option is useful only on Windows platforms.

9 Click the Non Blocking I/O check box to never block on a read or write when the file has an enforcement lock.

When another user has an enforcement lock (rather than an advisory lock) set on a file, if you try to read or write to that file, your application blocks on the read or write call until the other user unlocks the file.

This does not affect blocking on a lockFile() or lockSeg() method.

10 Click OK to build the I/O file object and return to the Browse dialog box.

Defining production objects

Production objects are created, modified, and tested in TRAN-IDE, not MSG-IDE. After the production object is created and tested in TRAN-IDE, it can be included in a MSG-IDE project. You can view the production objects included in a MSG-IDE project.

To viewing production objects in a MSG-IDE project, select Tools | Production from the menu bar.

Note For more information about TRAN-IDE functionality, see the *e-Biz Impact TRAN-IDE Guide*.

Defining timer objects

The timer object provides a means for setting an alarm in the application. Use it to execute a set of logic on a periodic basis.

Note Although the function associated with the timer is normally executed at the end of the interval set in the timer, there is an exception. If, at the end of this interval, some other process (such as a loop) is being run from within the AIM, the execution of the function associated with the timer will not occur until that other process has terminated. This is because the timer was designed to not allow its associated function to interrupt another process.

❖ **Defining a timer object**

1 Select Timer from the Classes list and click Add. The Timer window appears.

2 Click in the name Field and type the reference name for the timer object.

3 Click in the ID field and type an object ID value, which must fall in the range of 1 - 32767.

4 Click in the Interval field and type the number of milliseconds the timer should wait before performing the function identified in the Function drop-down list.

This is an initial value. This number is replaced by any value greater than 0 in the timer::set() method that makes the timer active.

5 Either click the Function drop-down list and select the name of the function the timer should initiate whenever it counts up to the time specified in the Interval box or click Details.

6 Choose one of the following:

- Click the One Shot check box to have the timer count up to the interval, perform the function, and stop. It never repeats the action unless you reset it with the timer::set() method.
- Do not click the One Shot check box to have the timer count up to the interval, performs the function, reset its counter, and restart the process. The only way to stop the timer action is to issue the kill() method from some other activity in your program

7 Click OK to complete your entries or modifications and build the timer.

Note See the *e-Biz Impact ODL Guide* for more information about the kill() and set() methods.

Understanding object flow

1 The communication object receives data from an application.

2 The Protocol Object receives the data from the communication object and places it into a blob.

3 The Protocol Object executes a Control Flow Object, if one is defined, to preview the data.

4 The Protocol Object then offers the data to its Message Frame Objects for bidding. Each Message Frame Object bids on the data based upon the position in the data that the Frame definition matches. The closer the Message Frame matches to the first position in the data, the lower its bid.

5 Decision Point:

- 5a. If no Message Frame Objects bid on the data, then the Protocol Object waits for another block of data from the communication object, appends it to the data currently in the blob, runs the Control Flow Object to preview the data, and submits the accumulated data to the Message Frame Objects for bidding.
- 5b. If Message Frame Objects match on the data, they submit a bid to the Protocol Object.

6 The Message Frame with the lowest bid (i.e., the one that matched closest to the first position in the data) gets the piece of the data that it matched and bid on. The Protocol Object extracts the matched data from its blob and passes that data to the Message Frame Object that won the bidding. The Protocol Object deletes from its blob any remaining data that came before the piece the Message Frame Object matched on and retains any data that came after the piece the Message Frame Object matched on.

7 After a Message Frame Object receives data from the Protocol Object, it passes program control to its Control Flow Object which executes its associated function.

8 Decision Point:

- 8a. If the Protocol Object has no data remaining in its blob, it waits for the next piece of data to arrive from the communication object, then starts the cycle over.
- 8b. If the Protocol Object still has data in its blob, it submits that data to the Message Frame Objects for bidding.

9 Decision Point:

- 9a. If no Message Frame Objects bid on the data, then the Protocol Object deletes the remaining data from its blob and waits for another block of data from the communication object.
- 9b. If Message Frame Objects match on the data, program flow follows as described in steps 5b – 8.