

eFramework™
PRACTICAL GUIDE TO WEB-ENABLING

Version 2.0



eFramework™ *Version 2.0*
Practical web-enabling using eFramework

COPYRIGHT

2000 by Integrated Systems & Services Group.

Manual No. eFW-FF-022

Printed in the United States of America.

All rights reserved.

August 2000

No part of this publication may be copied, or used in any form or by any means, without the express written permission of Integrated Systems & Services Group.

DISCLAIMER

Integrated Systems & Services Group has reviewed this manual thoroughly. All statements, technical information, and recommendations in this manual and in any guides or related documents are believed reliable, but the accuracy and completeness thereof are not guaranteed or warranted, and they are not intended to be, nor should they be understood to be, representations or warranties concerning the products described. Further, Integrated Systems & Services Group reserves the right to make changes to the information described in this manual at any time without notice and without obligation to notify any person of such changes.

TRADEMARKS

eFramework is a trademark of Integrated Systems & Services Group. Other product names mentioned in this manual may be a trademark or registered trademarks of their respective companies and are hereby acknowledged.

Contacting Integrated Systems & Services (ISS) Group.

Corporate Headquarters

Integrated Systems & Services Group
Overlook Tower
150 Clove Road
Little Falls, NJ 07424

Phone 973-812-9700
Fax 973-812-1460
Web site www.issgroup.net

Sales Department

Phone 800-955-6250
Fax 856-755-3659
Web site www.issgroup.net
e-mail address sales@issgroup.net

Technical Support Department

Phone 973-812-9700 Ext. 6009 or 877-HELP-028 (toll-free)
e-mail address helpdesk@issgroup.net

IV PRACTICAL GUIDE TO WEB-ENABLING USING EFRAMEWORK

Contents

- CHAPTER 1 OVERVIEW 7**

- CHAPTER 2 PROCESS SUMMARY 9**
 - Learning the product 9
 - Learn about product support 9
 - Pilot project 10
 - Pilot Review and Project Planning 11

- CHAPTER 3 TECHNICAL CONSIDERATIONS 13**
 - User Interface Design 13
 - Development environment 14
 - Fundamental web-enabling techniques 16
 - Row-level Security 20
 - Print Routines / Reports 21
 - Propath & Run Statements 23
 - Techniques for Managing Event Driven Code 23
 - Original Values 25
 - Knowledge Base 26

CHAPTER 1

Overview

Web-enabling a complex system is always difficult. e-Framework provides a strong set of technical tools designed to simplify this process. However, the technical tools must be used in conjunction with an overall implementation plan. This document describes the components of a standard e-Framework implementation. It also describes the best practice solution for specific implementation tasks.

Process summary

Learning the product

In order to use the e-Framework tool set effectively, a company should do the following:

- Identify appropriate technical resources and dedicate them to learning the tool set.
- Participate in a standard or customized five-day training class offered by ISSG. These classes cover the tool set in great detail.
- Upon completion of the formal class, spend one day reviewing all product documentation. E-Framework documentation includes a programming handbook, numerous tutorials, an installation manual and a reference manual. This step is extremely important. For example, many of the more advanced technical features of e-Framework are explained in detail in the reference manual.

In most cases, at least two technical resources should be assigned to learn the product.

Learn about product support

ISSG provides several methods for product support.

- Phone support.
- E-mail support.
- Integrated knowledge base.

The ISSG customer support process is designed to ensure that all support calls are managed accurately and expeditiously. Every call

originated by phone or e-mail is logged into our internal issues tracking system ("Caesar"). Each call is assigned a call number. Our internal support procedures mirror those of other organizations. The first line of support identifies whether they can handle the call. If not, the call is assigned to an e-Framework developer resource.

Please note that although it may seem quicker to get a response directly from an e-Framework developer, this will often take longer as calls that go outside the normal support channels are not managed by the help desk supervisor.

Pilot project

The first project to undertake should be tightly scoped to accomplish the following:

- One single-page transaction.
- One or two "standard" reports.
- Several lookups (five or more).
- One or two inquiries (lookups with filter criteria).
- One multi-page transaction (e.g. simplified order entry). Note that the pilot project should contemplate two or three page transactions only.

These individual tasks should be implemented in the order listed (i.e. "easiest" to "hardest").

This pilot can be executed by ISSG as part of the eAdapt e-Framework implementation methodology or by the client themselves. In the latter case, as the pilot is conducted, developers should frequently consult with ISSG.

The goals of the pilot are:

- Identify standards to be used for future development.
- Become familiar with the e-Framework tool set.
- Become familiar and comfortable with ISSG technical support.
- Identify potential shortcomings in the e-Framework tool set and work with ISSG to either enhance the tool set or determine appropriate work-arounds.

Pilot Review and Project Planning

Review the pilot at both functional and technical level. During this review process, a representative from ISSG, the client's technical resources and client management should discuss every aspect of the pilot.

Technical considerations

This section describes a set of problems that must be addressed and resolved before and during the pilot. A full project should not begin until each of these has been addressed and understood.

User Interface Design

The user interface usually mirrors that of an existing web site. The standards used to develop the corporate web site should be documented and that documentation should be made available to the developers.

Note that in some cases, the user interface will be developed by a separate set of resources or even out-sourced.

In all cases, e-Framework developers should modify the Web Object Generator template files (wogtemplate.htm, wogtemplatem.p, wogtemplates.p and wogtemplate.w). These files are described in detail in the Reference Manual. Developers modify these files so that the user interface of new functions created by e-Framework is "close" to the corporate standard. In most cases, just wogtemplate.htm will be modified. In some cases, the other files may also change. Consult with ISSG in all cases.

In addition to modifying the template files, the client should select a WYSIWIG ("what you see is what you get") authoring tool such as Macromedia's Dreamweaver.

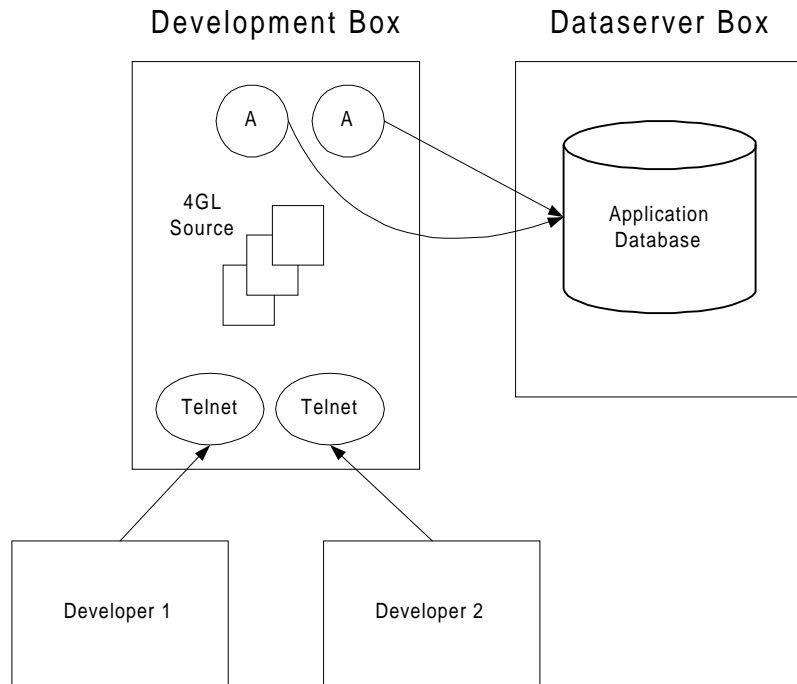
Development environment

Development using WebSpeed is different that development using the Progress UIB/ADE. Code executes on machines co-resident with the WebSpeed agents.

Developers accustomed to executing and testing code on personal computers must adjust to this fact.

There are many different development scenarios, as shown in the diagrams below.

Fig. 3.1 Host-based development environment



WebSpeed Agents (denoted by "A" in the diagram) are always co-resident with the 4GL code. They connect to the application database either over the network (as depicted) or directly via shared memory.

Developers connect to the development box via a telnet client and use editing tools available on that box to edit source code.

For convenience, these telnet sessions should execute a 4GL environment configured with a complete propath and appropriate database connection. Developers can then easily check syntax on code before executing those programs over the web.

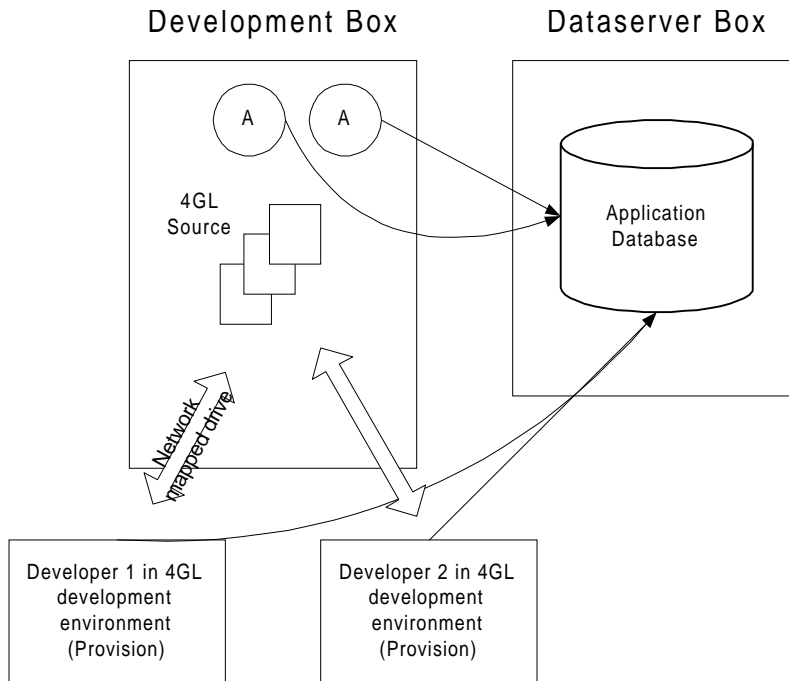


Fig. 3.2 Client-server based development

Developers use Windows machines running Progress Provision (or other appropriate software). They map network drives to the development box and do all editing using the Progress Windows editor. The Progress client should configure appropriate PROPATH and database connections.

This alternative is recommended, especially when the Progress client is version 9 or higher as the GUI editor is very powerful.

In addition to source code location, it is also important to realize that all agents for a given WebSpeed service use the same Progress propath. This complicates a multi-developer environment. Assume a project is assigned two developers. Developer A works on program a.p. Developer B works on b.p. Both these programs reference a common include file, {stdinclude.i}. If developer B makes a change to the include file, developer A will be affected by the change.

Fundamental web-enabling techniques

Applications vary widely from organization to organization. However, ISSG has identified certain standard transformations made to web-enable 4GL source code. They are described in the following sections.

Statements requiring input from the user

Consider the following program fragment:

```
repeat:
  prompt-for dbtable.dbfield.
  find dbtable where
    dbtable.dbfield eq input dbtable.dbfield.
  update [a list of dbfields in dbtable].
end.
```

Note The find statement could also be implemented as:

```
Find dbtable using dbtable.dbfield.
```

Apply the following transformation rules to the above:

- Replace the UI statement (prompt-for, update, set) with a call to an appropriate GetValue (iGetValue, decGetValue, etc) for each associated field.
- Remove actual prompting 4GL statement (update, set, prompt-for) from the code.

In addition make a note of each field. These fields must be present on the HTML user interface. Once the entire program has been analyzed, create one or more web objects that define these fields.

Note

- iGetValue is used for integer inputs.
- decGetValue is used for decimal inputs.
- lGetValue is used for logical inputs.
- dtGetValue is used for date inputs.
- cGetValue is used for character inputs.

Statements that display on the UI

Consider this code fragment:

```
repeat:
    prompt-for dbtable.dbfield.
    find dbtable where
        dbtable.dbfield eq input dbtable.dbfield.
    if available dbtable then
        display [one or more dbfields, variables and/or expressions]
    update [a list of dbfields in dbtable].
end.
```

The above example is similar to the previous except with the addition of the code in bold. Transform the code as follows:

- 1 Convert each dbfield displayed on the screen to an appropriate PutValue function call. For example, if dbfield is an integer, use iPutValue or if dbfield is a decimal, use decPutValue and so on.
- 2 If the output is not going to the unnamed stream or it is going to the unnamed stream but the unnamed stream's output has been re-directed elsewhere, ignore this particular display statement altogether. This is a very important exception to this rule because report programs typically re-direct the unnamed stream to a file or printer and then use the display statement to display the actual report content. Note that this can also be said of update statements.

As with prompting UI statements, displayed fields must be tracked and entered into the web object generator.

Standard 4GL error handling

Consider this code fragment:

```

repeat:
  prompt-for dbtable.dbfield.
  find dbtable where
    dbtable.dbfield eq input dbtable.dbfield.
  if available dbtable then
    display [one or more dbfields, variables and/or expressions]
  else do:
    message "ERROR: Record not on file.".
    leave.
  end.
  update [a list of dbfields in dbtable].
end.

```

The above example is similar to the previous except with the addition of the code in bold.

Note

- The "leave" statement may be a "retry" statement.
- The "leave" statement may be a "next" statement.

Transform the code as follows:

1 Convert warning type messages to:

```
lSetWarning(-999,"[message statement's expression]").
```

2 The message statement above would be transformed into:

```
lSetWarning(-999,"ERROR: Record not on file.").
```

In the case of an error message, use `lsetError()` instead.

3 Any leave, retry or next statement is converted to a return statement.

Note Messages should be defined using Application Messages Maintenance. It is bad practice to use message number -999.

Frame Validations

Progress allows field level validations to be defined in two places:

- Data dictionary.
- 4GL code in frame definitions.

At compile-time, the progress compiler builds "frames" and stores all frame related information in p-code that is accessed by the run-time engine to enforce these validations.

Although there is a frame in the WebSpeed environment, it cannot be used to apply frame-level validations. There are several possible approaches to this. The most straightforward is to implement one or more validation procedures to apply validations in response to UIE events (as defined in Event Master Maintenance). This approach results in duplicated validation code.

It is also possible to apply validations at run-time using existing database field and frame validations. This is an advanced technique and involves "walking the widget tree". Contact ISSG for additional information on this approach. Future versions of e-Framework may supply low-level tools to facilitate this type of validation automatically.

Include Files

There are two types of include files to consider: include files that interact directly with the user interface and include files that do not. In general, the latter type will function correctly without any changes. UI-related include files must be analyzed and transformed according the rules listed in this section.

When modifying one of the UI-related include files, consider enhancing it so that it can work in and out of a web environment. If this cannot be done, the include file must be cloned and modified. Every occurrence of the include file must be replaced with a reference to the new include file.

Repeat Loops

Any repeat block in which the user is required to supply input must be converted into a "do" block instead.

Temporary Tables and Stateless Applications

E-Framework applications are always stateless. Applications that use temporary tables (and workfiles) usually expect those tables' existence to span multiple user interface activities. In a web environment, this is not the case.

Follow these steps to use a temp-table in an e-Framework application:

- 1 Create a temporary ASCII file using "cNewTempFile()" defined in fileutil.i. Store the name of that temp-table in an e-Framework session variable using lAddSessValue() and lPutSessValue().
- 2 Once an application process step has finished using the temporary table, store its contents into the ASCII file. ("for each tt: export tt. End.")
- 3 Re-create the temporary table before executing any validation or processing step that requires its data. Get the name of the ASCII file using cGetSessValue().
- 4 Once the temporary table is no longer needed, use iDeleteFile() defined in fileutil.i to delete the ASCII file.

Row-level Security

This is one of the most important technical problems and functional issues that must be understood and solved in order to implement and system ready for production. Row-level security is the type of security that governs access to rows within a table. This is separate from the more common application security such as menu access or field access within a function.

There are two general cases to consider:

- 1 Transactional. Transactional functions are relatively simple to secure. When a form is submitted, the application applies security rules to the submitted data. This is best accomplished using the web object generator (click on the "security" button and check off which fields should be secured using security rules). Alternatively, transactional functions can also be secured by implementing security code as validation events in the X file.
- 2 Inquiry. Inquiries are more difficult technically. This is due to the fact that they are real-time and require real-time performance. It may be necessary to add additional indexes to some database tables in order to implemented efficient security lookups.

In both cases, the specific e-Framework functions and maintenance utilities to configure security are documented in great detail in the standard e-Framework documentation. Review "Lookup Tool" in both the e-Framework Reference Manual and the Programming Handbook.

While considering the technical aspects of row-level security, organizations must also consider the functional requirements. E-Framework provides a "grant" security model that is based upon certain pre-defined rules. In a grant model, security access must be granted explicitly for each piece of secured data. Many applications implement an exception based security model. For example, all users may access function "X" except users ID's beginning with "admin". Note that ISSG has documented a technique for defining an exception-based model as well. Refer to the on-line knowledge base or contact technical support for details.

Furthermore, e-Framework is shipped with a handful of sample application rules. They are:

- Customer access.
- Site access.
- Item access.
- System access.

These application rules are backed up with a set of maintenance utilities and 4GL functions to implement them in an application. These rules were developed to support an existing ERP system. However, they can be adapted for use in other environments. Contact ISSG for assistance in these cases.

Print Routines / Reports

There are several technical approaches to web-enabling reports. They are:

- 1 Use I/O redirection. Create a simple web front end that captures report criteria. The X file creates an ASCII file in Progress export/import format. It then changes default input to come from that ASCII file and redirects output to another ASCII file. The report is executed and the result ASCII file is displayed onto the screen.
- 2 Re-engineer existing reports at the 4GL source code level.
- 3 Integrate with 3rd party formatting tools.

The following table lists important factors for each of these approaches.

Technical approach	Benefits	Problems
I/O Redirection	Least development effort.	No integrated security. "Ugly" report output not usually suitable for a web audience.
Re-engineer 4GL source code	Can generate excellent web "Easy" security integration.	Potentially substantial development effort.
3rd party tools (e.g. Optio, StreamServe)	Dedicated product to creating excellent HTML interfaces. Can be leveraged in other (non-web) areas of business.	No integrated security. Must be trained on new tool. Cost of tool.

ISSG recommendations:

- Use I/O Redirection for those functions used by employees. Security issues are usually less important and employees are already accustomed to standard report output format and definitions.
- Re-engineer 4GL source code: Incorporate appropriate security for customer (non-employee) access. However, avoid modifying report output to conform to web requirements. Instead, utilize a 3rd party formatting tool to accomplish this.
- Use 3rd party tools to change report output for display purposes only. This is easier to justify when there are a relatively large number of reports to web-enable (10 or more but possibly less).

- Aside from technical matters, it is important to consider whether a particular report should be web-enabled at all. In many cases, it is better to provide an inquiry function using the e-Framework lookup tool in place of an existing report.

Propath & Run Statements

There are many different methods for running 4GL programs. Applications often utilize a set of standardized include files to execute 4GL programs. These include files may have to be modified in order to operate properly in the WebSpeed environment.

In addition, e-Framework provides a standard solution to handle multi-language applications. One use for standardized include files is to automatically manage multi-language issues. This is a specific area that must be addressed during the pilot.

Techniques for Managing Event Driven Code

Note This section is also generally applicable to the use of the "readkey" statement for character-based applications.

Starting with version 7, PSC introduced an event driven model and 4GL statements to support that model. In addition, the GUI tool set provided by PSC (UIB, ADE or AB for versions 7, 8 and 9 respectively) guide programmers into an event driven programming paradigm.

Most event driven programming focuses on the user interface. It is common to apply field-level validations, hide, enable or disable fields, change UI characteristics, etc using 4GL triggers.

This type of field-level validation cannot be implemented in the same way in a web environment. Standard HTML does not provide for any field-level events. It is possible to implement some field-level events using JavaScript and this can be very effective in some situations. However, any validation that requires access to a database table must be done on the server side (as opposed to the HTML client side). This is accomplished either through a direct post of the form or through the use of a hidden "trampoline" frame.

In e-Framework development, standard event-driven field validations should be implemented as follows:

- Data type checking: Specify data types of fields in the web object generator. When a form is submitted, the e-Framework run-time engine validates the data type of each field vs. the submitted value. In the event of a data type checking error, the user is notified and no further processing takes place.
- Field level security (existing security rules): Again, use the web object generator to specify which rules should be applied to data entry fields.
- Custom field level security (no existing data access rule applies): Write a validation event for the form.
- Enable/disable fields unconditionally: This means that some fields are always display-only (e.g. h_error and h_status). Use ip-disable-field or ip-hide-field in the supplemental-context program ("S file").
- Conditional enable-disable of fields: Use JavaScript, often in conjunction with hidden fields. In some cases, modify the s-file.
- Auto-fill fields: Use JavaScript, often in conjunction with hidden fields.

In addition to field-level validations ("on leave" triggers and so on), the event driven model is used for application buttons (e.g. save, delete, help, etc). For these types of events, use e-Framework event master maintenance to create a UIE (user interface event), one or more validation events and one or more processing events.

Cursor up/down function is commonly used to simplify the data entry process. For example, a data entry screen may prompt for a sales order number. When the order number field has focus, users can often press cursor-up to view a previous order and cursor-down to view the next order. A browse may be associated with the field. E-Framework implements cursor-up/cursor-down functionality with a standard include file, wdmnav.i. Read the e-Framework reference manual for details. Note that the web object generator can be used to automatically include next/previous logic. Use the e-Framework lookup tool to create browses and link these browses to the data entry form via Event Master Maintenance.

Original Values

Given the stateless nature of the Internet, e-Framework applications cannot rely upon the 4GL's standard record locking mechanism to guarantee exclusive access to data during a particular data entry function.

E-Framework provides tools to allow developers to implement an optimistic locking strategy. The goal of this method is to detect collisions rather than prevent them.

Consider the following non-web locking strategy:

```
prompt-for customer.nbr.  
find customer using nbr  
    exclusive-lock.  
update customer except nbr.
```

When this program is executed, the user enters customer number 100 and hits the update statement, customer number 100 is now locked. If another user executes the same program, that user cannot access customer 100.

The equivalent web program is as follows:

```
Find customer where customer.nbr = cGetValue("h_customer_nbr")  
    exclusive-lock.  
customer.field1 = cGetValue()  
customer.field2 = cGetvalue()
```

In a stateless environment, two different users can access the data entry function. User A can access the function and see the customer's current data. User B does the same. User A walks away and gets a cup of coffee. User B enters some data and presses submit. Later, use A returns, keys in some information for customer 100 and submits the form. This will overwrite user B's changes.

In some cases, the above is desirable. In most cases, it is not. In the host-mode or client-server environment, this could not have happened because user B could never have reached the update statement.

The e-Framework solves this problem by detecting the fact that user A's changes are based upon old data and may no longer be valid. This is accomplished with the following logic:

- When user A access the function, a current snapshot of the data is stored into the e-Framework context.
- User B accesses the function. As with user A, a current snapshot of the data is stored into the e-Framework context.
- User B makes a change and save it.
- User A makes a change and submits it. E-Framework compares the current snapshot picture to the stored snapshot picture. If they are not the same, a collision has been detected and user A is informed.

Use the web object generator and "Original Values Maintenance" to configure this optimistic locking strategy. The technical implementation of this solution is described in the e-Framework Reference Manual.

Knowledge Base

E-Framework is shipped with a knowledge base application and data. The supplied data consists of common techniques, tricks and work-arounds to e-Framework and general web programming issues.

The KB is referenced by most e-Framework error messages.

New applications created using e-Framework should usually incorporate the knowledge base as well.