# Top Five Myths of Screen Scraping

## The Evolution of Application Modernization Technology

A WHITE PAPER FROM

Rocket | Seagull™

www.rocketsoftware.com/seagull

Imagine for a moment it's your first day at a new job. You're working for a company that relies on timely and accurate access to customer information. Perhaps you are a claims processor at an insurance company, a customer service rep at a bank, or a sales manager at a car dealership. One of the first tasks your supervisor gives you is to spend your first three weeks on the job learning a business-critical computer application that looks like it's from 1975. How would that make you feel?

Surprisingly, many organizations still ask their employees to use green-screen (or text-based) user interfaces to work with essential business information. If your organization uses or sells mainframe, System i or OpenVMS applications, you're familiar with this scenario. And if you manage employees who use these applications, you recognize the problem green-screens present. They're unappealing, hard to learn, intimidating, and accessible only with terminal emulation software. They're also incompatible with modern application integration standards like Web services and SOA.

Green-screens are a business problem and a technology problem. How can businesses stay competitive when it requires weeks to train employees on green-screen applications? How can a group of customer service reps help a customer on the phone when it takes several minutes to find basic customer data? How can a business share select pieces of critical data with employees, partners, and other applications without giving users a complicated way of getting to the data?

The technology side of the problem is simply that green-screen applications are no longer the de facto standard for user interfaces. Host-based applications run businesses. They're still around today because they're reliable, secure and have decades of business rules and information housed in them. Application developers need a way to help the business use the same information from the legacy system but deliver it in a modern and flexible way.

## Getting Past the Myths

For several years there have been ways of turning green-screen applications into more up-to-date and useful Web-based applications or services. This approach is commonly referred to as *screen scraping.*

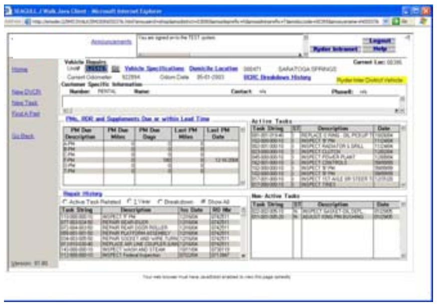
Screen scraping has negative connotations to it — some consider it a quick fix and a clumsy way to develop applications or services. But this attitude is out of step with today's application modernization technology. With the right combination of features, a screen-scraping tool can be an application modernization solution that solves the business and technology problems that green-screens present.

This purpose of this white paper is to outline the most common screen-scraping myths that proliferate today and show how current modernization technology challenges those myths.

## Myth #1: Screen Scraping is Only a Green-Screen in a Browser

A basic requirement for green-screen-based modernization is the ability to dynamically display a GUI version of any application screen. This typically involves setting up generic templates or rules that interpret each screen on the fly and present a GUI version of the screen to the user. Dynamic GUI is useful for deploying applications to desktops or the Web, but adds little functional value from the user or business perspective. It is sometimes referred to as "green-screen in a browser."

The real value-add in screen-based modernization projects lies in the ability to customize the application user interface. Users need improved application usability and workflow and integration with the desktop- and Web-based tools they require to do their jobs. The basis for a customized GUI is still screens; however, the screens behave more as application-level APIs than direct user interfaces.



Customized interfaces give users the same host information with much improved functionality.

Traditional screen-scraping approaches focus on "green screen in a browser." Current modernization technology focuses on re-engineering the application at the user interface level, combining information from multiple green-screens and presenting it to the user when and where it is required to perform whatever business processes they are engaged in.

## Myth #2: Keeping Host and GUI in Sync is Hard

As any developer knows, maintaining APIs can be a labor-intensive process. If an API changes, anything that uses that API must typically also be adapted. So if a green screen changes, how do you take that change into account in your GUI?

Managing host application change is a key differentiator between current modernization technologies and a typical screen-scraping approach. An effective modernization solution uses your actual application screen maps[1] during application development and maintenance. With a repository of screen maps as a basis for development, developers can create and maintain customized GUIs for applications that include thousands of screens. When screens change, an automated change management feature enables you to easily synchronize your GUI application with your underlying screen "APIs."

Host-to-GUI synchronization is actually a two-part challenge:

- The GUI must know what screen(s) to use to access information
- The GUI must know what host fields to use on each screen

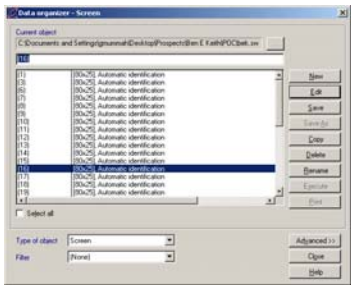
[1] Mainframe applications typically use screen map types like BMS and MFS. System i applications typically use display files (DSPF).

Traditional screen-scraping approaches expect you to manually maintain the links between your live green screens and your GUI. If a screen changes, there is no automated way to catch and apply the change. By registering your application screen map files in an XML-based repository, today's modernization tool overcomes the synchronization challenge and provides a complete foundation for application development and maintenance.

## What screen am I on?

Custom GUIs for green-screen applications require a link between the GUI and the host screens that provide information. In other words, the GUI solution must "know" what screens are being sent by the host in order to show the right GUIs.

A stereotypical screen-scraping product provides manual screen identification. In manual identification, the developer must navigate to each desired screen in the live application and manually select the characteristics about that screen that make it unique from all other screens in the application. If the developer accidentally selects a conditional field or uses the same characteristics to identify two different screens, identification can fail. If a screen changes, the developer must manually navigate to the screen again and re-identify it.



Current object
C:\Documents and Settings\gmummah\Desktop\Prospects\Ben E Keith\POC\bek.sw
[16]

```
[1]   [80x25], Automatic identification
[3]   [80x25], Automatic identification
[6]   [80x25], Automatic identification
[7]   [80x25], Automatic identification
[8]   [80x25], Automatic identification
[9]   [80x25], Automatic identification
[10]  [80x25], Automatic identification
[11]  [80x25], Automatic identification
[12]  [80x25], Automatic identification
[13]  [80x25], Automatic identification
[14]  [80x25], Automatic identification
[15]  [80x25], Automatic identification
[16]  [80x25], Automatic identification
[17]  [80x25], Automatic identification
[18]  [80x25], Automatic identification
[19]  [80x25], Automatic identification
```

Select all
Type of object: Screen
Filter: (None)

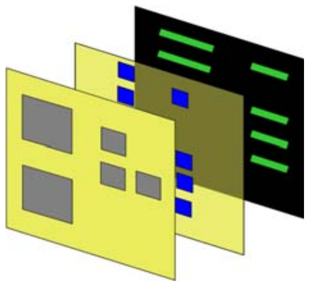
As shown in this repository, being able to keep track of screens is a key difference between screen-scraping and modernizing.

With a more advanced modernization tool, maintaining screen identifications is an automated process. Screen maps are incorporated into a repository and are automatically assigned unique identifiers. Since this process is based on the screen maps rather than the live datastream, it's more accurate. The links between GUIs and the green-screens in the repository are automatically checked and updated whenever the change management process is initiated.

Automated screen identification insures an accurate link between your GUI and your host applications – without requiring a developer to spend any time maintaining these critical links.

## What fields do I need?

As users work with your GUI, information they enter or change is pushed back to editable fields on the green screen. Therefore, GUIs built over green-screen applications require a link between the host fields on each screen that provide the information and GUI fields that enable users to interact with that information.



Rather than using row/column positions, an effective Web-enablement tool should offer automatic host field mapping.

A typical screen-scraping approach to this challenge is to rely on the developer to create manual mappings based on the row/column position of each host field on each screen. Developers navigate live to each screen that contains the fields they want, then point and click on each individual field to register it for use in the GUI. Solution developers may not know that certain fields change color, size, or position based on program events and therefore cannot take these changes into account. They may be unaware of conditional fields that only display in certain circumstances. Without access to the screen maps, these solutions place the burden on the developer to discover changes in field states and attributes.

A more precise method is necessary. An effective modernization tool offers automatic host field mapping based on the application field names and attributes from the screen maps. Information about every host field in the application becomes available via the screen repository. By providing access to this information, the solution enables developers to create solutions that encompass all of the host application possibilities — including features like detecting and acting upon color and state changes and variable field positions and sizes.

Using the true application field names also reduces change management requirements, as the modernization solution interacts with fields based upon their true host names rather than their row and column positions on the screen. If you move a field on the screen, a modern solution can still interact with it without changing the underlying field definition.

## Myth #3: Screen-Scraping is a Maintenance Nightmare

One of the most pervasive screen-scraping myths is that it causes a maintenance nightmare. This myth is well-earned, because in traditional screen scraping solutions maintenance is a challenge. Because these solutions scrape live screens for development, they offer no automated way to accommodate for, or even be aware of, host application changes. This shifts the maintenance burden to the developer, who is expected to identify and react to these changes. This reactive approach often means that maintenance is not initiated until *after* users have already found a problem.

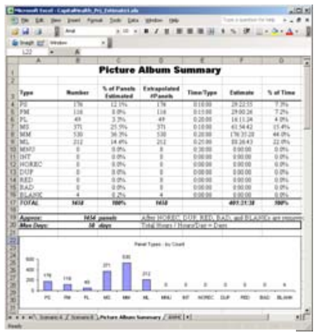
As already stated, a modernization tool should feature change management technology designed to keep application screens and corresponding GUIs synchronized. This point is critical. Accurate change management reduces your application maintenance burden and elevates the quality and reliability of your modernized solution.

When a solution uses a screen repository to manage host-to-GUI screen changes, ongoing maintenance becomes a highly automated process. Developers can proactively schedule synchronization with the host application screens at any time. The synchronization process updates the screen repository with the latest screen maps, makes corresponding updates to screen identifications, host field information, and the GUI, and produces a complete report of all activities. These processes can all run unattended (in batch mode), or they can be performed step-by-step by individual developers. Work is typically limited to adjusting the layout of the customized GUI to accommodate added fields on the host and customizing GUIs for new host screens. Working files can also be managed via common source control tools like CVS or Subversion.

## Myth #4: Screen-Scraping Projects Are Unmanageable

Screen scraping is often seen as a quick fix to complete a project at the last minute. Because of this perception, project management is often not taken into account until after a solution is identified. This can be a costly mistake. With the traditional screen-scraping approach, developers operate in the dark. When building a new GUI front-end solution based only on live screens, the visibility into the level of effort is limited to the developer's knowledge of the live application. You may be unaware of application features that only display in certain circumstances, or of modifications that only apply to certain customers or business units. Your visibility into maintenance costs is also limited.

However, if you're using a modernization solution that provides you with complete information about all of the screens that are part of your project, you'll be able to clearly understand, estimate, and document the amount of work required to complete a modernization project. If you are considering a screen-based Web enablement project, you certainly have applications in mind that you want to convert. How many screens are in those applications? If you don't know how many screens there are to create GUIs for, how can you estimate the time and effort it will take to complete the project? How do you even know when the project is complete?



Picture Album Summary

| Type | Number | % of Panels Estimated | Extrapolated #Panels | Time/Type | Estimate | % of Time |
|---|---|---|---|---|---|---|
| PS | 176 | 12.1% | 176 | 0:10:00 | [illegible] | 7.3% |
| PM | 118 | 8.0% | 118 | 0:15:00 | [illegible] | 7.2% |
| PL | 49 | 3.3% | 49 | 0:20:00 | 16:11:24 | 4.0% |
| MS | 371 | 25.5% | 371 | 0:10:00 | 61:54:42 | 15.4% |
| MM | 530 | 36.3% | 530 | 0:20:00 | [illegible] | 44.0% |
| ML | 212 | 14.6% | 212 | 0:25:00 | [illegible] | 22.0% |
| MMU | 0 | 0.0% | 0 | 0:30:00 | 0:00:00 | 0.0% |
| INT | 0 | 0.0% | 0 | 0:00:00 | 0:00:00 | 0.0% |
| HOREC | 0 | 0.0% | 0 | 0:00:00 | 0:00:00 | 0.0% |
| DUP | 0 | 0.0% | 0 | 0:00:00 | 0:00:00 | 0.0% |
| RED | 0 | 0.0% | 0 | 0:00:00 | 0:00:00 | 0.0% |
| BAD | 0 | 0.0% | 0 | 0:00:00 | 0:00:00 | 0.0% |
| BLANK | 4 | [illegible] | 4 | 0:00:00 | 0:00:00 | 0.0% |
| TOTAL | [illegible] | 100% | [illegible] |  | [illegible] | 100% |

An effective modernization solution gives you visibility into your project.

By using a screen repository, a current modernization solution lets you quantify the work required to complete a project. Tools should exist in the solution that let you ascertain the exact number of screens in any application or module, as well as the *complexity* of each screen. A project estimator feature enables you to assign time-to-complete values for screens of varying complexity and document the level of effort, down to individual screens and fields, to successfully implement your GUI interface.

## Myth #5: You Can't do SOA with Screen-Scraping

Thus far this paper has focused on new graphical user interfaces for existing green-screen applications. Some screen-scraping solutions also offer the ability to service-enable existing applications. Service enablement means taking an existing application function, like "look up a customer address" or "add an item to an order," and encapsulating it into a callable service (typically a web service). The service is then made available to other applications as part of a larger service-oriented architecture (SOA) project. When other applications need information from the host application, they call the required application functions and receive a reply via web services. The underlying host application remains unchanged.

While this is a different type of modernization for an existing application, all of the challenges that apply to creating a new GUI also apply to creating new services. Traditional screen scraping approaches still don't address the fundamental challenges — automating service mapping to the right screens and fields, synchronizing host changes with the services that rely on these screens and fields, and being able to accurately estimate the level of effort required to create and maintain services. Screen-scraping is also traditionally thought to be too slow to meet the service level expectations of a modern SOA. This is typically because traditional service-enablement functionality is bolted on to an existing screen scraper-based GUI enablement solution rather than executed separately.

In order for a service enablement solution to work in high-volume environments that require frequent changes to business logic, it must be capable of busting all of the myths presented in this white paper. It also should execute services *independently* of any GUI modernization layer to insure maximum performance.

## LegaSuite: Painless Screen-Based Modernization and Integration

If you're looking for screen-scraping solution but are wary of the development hurdles or the quality of the result, LegaSuite will change your outlook. LegaSuite goes beyond screen-scraping to offer a complete Web- and service-enablement solution that can modernize your existing text-based application portfolio.

If you're planning a Web- or service-enablement project for an existing host-based application — whether it's based on the mainframe, System i, OpenVMS or UNIX-VT platform – consider the advantages LegaSuite offers:

- Synchronized development and maintenance for screens and their corresponding GUIs and services
- Automated screen identification and management
- Accurate host-to-GUI and host-to-service field mapping based on true field names instead of row/column positioning
- Detailed project planning and management capabilities

Whether you have used another solution in the past or are just beginning to explore the possibilities of Web- and service-enablement for your existing applications, we invite you to experience the LegaSuite advantage. Contact us today for a no-obligation, personal demonstration.

**About Rocket Seagull**

Since 1990, Rocket Seagull Seagull has specialized in technology that lets organizations quickly and easily reuse the information and data housed in their legacy systems.

Over 1,800 customers and 350 ISVs use our LegaSuite software platform to connect legacy applications on mainframe, IBM i (AS/400), OpenVMS, and Windows client/server platforms to the Web, to other middleware and to newer-generations of applications.

www.rocketsoftware.com/seagull