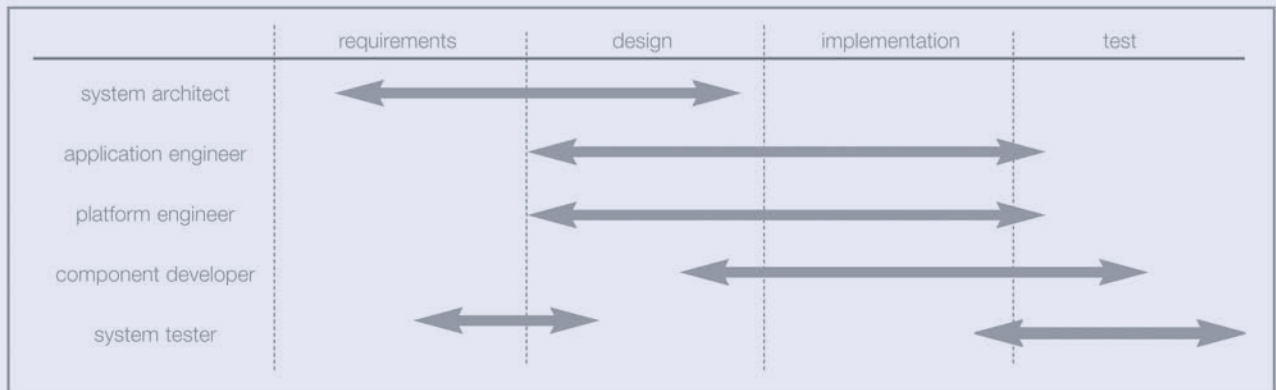
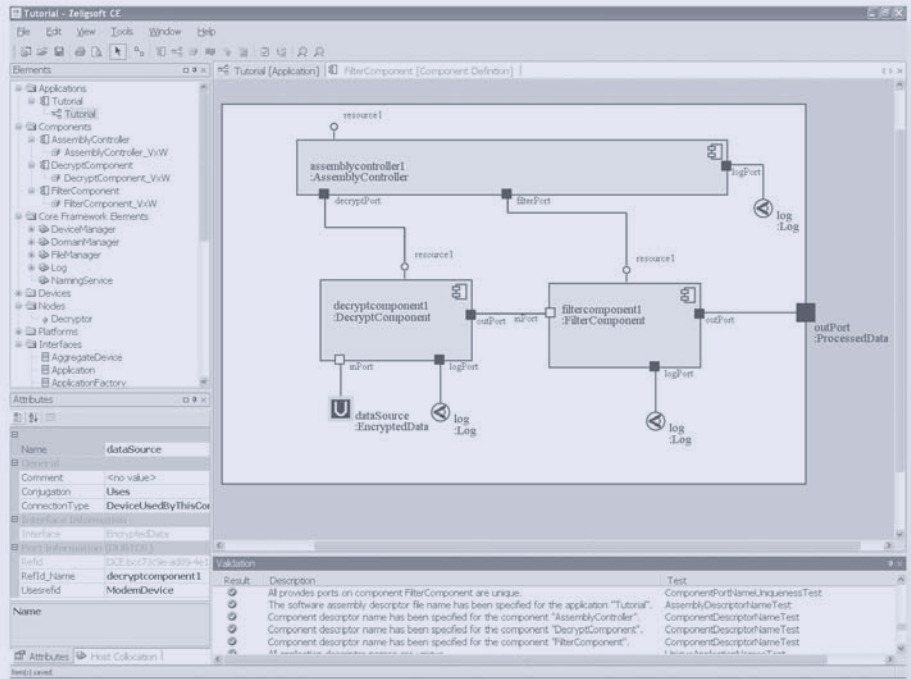




Validating SCA Compliant Systems:

SCA Descriptor Validation using
Zeligsoft Component Enabler

Mark Hermeling, Francis Bordeleau



Validating SCA Compliant Systems: SCA Descriptor Validation using Zeligsoft Component Enabler

Mark Hermeling, Francis Bordeleau

Abstract: All project teams developing software based on the Software Communications Architecture (SCA) standard need to create descriptor files that accurately describe their system. Many teams have manually authored these descriptor files early in their development process. These files are then used during unit testing, integration, system testing, and in real-world deployment of the system. However, using existing methods does not properly validate these descriptor files. This leads to a development process that is more expensive than necessary, as well as possible project and budget overruns due to unexpected complications during the system integration process. This paper discusses the cost of unvalidated descriptor files and provides a tool-based solution. The solution reduces risk and increases predictability of SCA-based software development projects by validating the descriptor files early in the development process.

1 INTRODUCTION

Component-based systems are based on combining individual components together into a complete system. This means that all the components need to be synchronized. Their interfaces need to be compatible so that they can work together. A component-based system also needs some form of description of how the components are assembled. These descriptions are mission critical artefacts. They reflect the composition of the system, how the system is deployed to hardware, and how to configure the system for different scenarios. An error in the system description can render the system faulty or otherwise inoperable.

This paper examines the system description language for the Software Communications Architecture (SCA), the standard defined by the Joint Tactical Radio System (JTRS) task group of the US Department of Defence.

We cover the system description, also known as Domain Profile, for this standard. The Domain Profile for the SCA consists of a set of XML descriptor files that are normative and have to adhere to a strict syntax. The complete Domain Profile contains the information that is needed to successfully download, deploy, execute, and maintain the system.

The descriptor files used in the SCA are complex. Authoring and maintaining these files is a significant undertaking that requires a high level of expertise in the standard. Once authored, the files are difficult to read and navigate. Validation of the descriptor files requires domain-specific tools that are tailored to the rules and regulations specified in the standard.

This paper looks at validating the descriptor files. We make the case that unvalidated descriptor files are a liability: they create problems at multiple stages in the development process for SCA-compliant systems.

We introduce Zeligsoft Component Enabler (CE), a software modeling tool that provides validation and visualization capabilities. CE allows users to locate — and help to resolve — errors in the descriptor files. Correct descriptor files provide a stable reference point in the development of SCA-compliant systems. They prevent errors and misunderstandings early in the development process and make SCA-based software development more predictable.

Section 2 introduces the problem statement. Section 3 explains existing validation methods, which are in use by most SCA-based projects. Section 4 introduces and classifies the errors that can occur in descriptor files. The classification also provides information about how

existing methods validate errors, as well as the impact that these errors can have on the behavior of SCA-compliant systems. Section 5 introduces Zeligsoft Component Enabler and briefly describes its capabilities. Section 6 describes CE's XML Import feature, which provides information on how CE helps in the validation of descriptor files. Section 7 provides a summary.

This paper assumes that users have a basic understanding of the SCA and the descriptor files mandated by the standard.

2 PROBLEM STATEMENT

The SCA mandates the use of the eXtensible Markup Language (XML) in their descriptor files. These descriptor files must adhere to a specific syntax as defined in the Document Type Description (DTD).

The descriptor files comprise the Domain Profile, which defines an entire system. The Domain Profile consists of the Software Profile (the description of the software), and the Device Profile (the description of the hardware), and device drivers. The files in the Domain Profile include the following:

- **Software Assembly Descriptor (SAD)**
contains the description of an application as an assembly of instances of components.
- **Device Configuration Descriptor (DCD)**
is similar to the SAD, but it defines a hardware board as an assembly of device instances.
- **Software Package Descriptor (SPD)**
contains the definition of a component or a device, as well as information about their implementations.
- **Software Component Descriptor (SCD)**
contains the definition of the external interface of a component or device.

- **Properties Descriptor (PRF)**

contains information on the properties of a component or device. The PRF can be referenced through the SPD or SCD. The SAD and DCD can override properties defined in the PRF.

- **Domain Manager Configuration Descriptor (DMD)**

contains information on the initial services for the Domain Manager.

- **Device Package Definition (DPD)**

contains revision and author information about physical hardware.

The XML in the descriptor files is not easily human-readable or -writable. The syntax is complex and contains many cross-references, within and between descriptor files. These references are required because the descriptor files need to describe an entire system as an assembled collection of individual components. SCA-based systems can be rather large in size, often totaling 10 to 60 thousand lines or more of XML code. Manually writing this much XML is painful, expensive, and error-prone — especially considering the size and the number of cross-references. Certainly, development time can be better spent.

Many current SCA development projects debug the XML during run-time. The SCA Core Framework (CF) reads the XML and provides feedback; however, the feedback is often very cryptic and not easily interpreted. Moreover, the CF does not use all of the XML, so feedback is limited. Every test-run requires a significant amount of setup time, which again is painful and expensive. Like an application that compiles, the fact that the XML starts an application seemingly correct is not necessarily proof that it is completely correct. Other errors can pop up during run-time, when the XML is migrated to a different platform, or when parts of the system are re-used in another project.

Current projects develop an architecture on paper, then independently develop the descriptor files and the code to suit this architecture. Validation of the descriptors

(and hence the architecture) is only possible during software integration, once enough code and descriptors have been written that the application can be executed within a CF. This means that precious time has been spent developing against an unvalidated architecture. This can create significant problems in the later stages of the software development process.

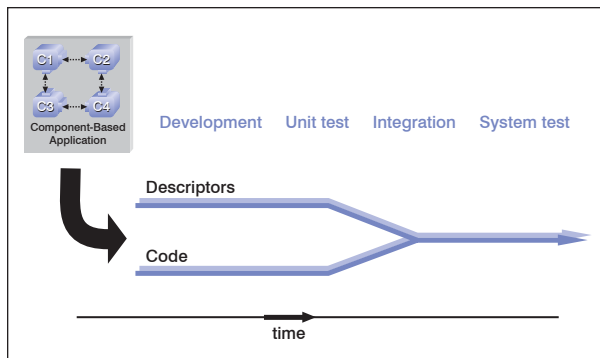


Figure A — Software Development in the SCA

Figure A shows the process that many projects follow when developing software for SCA-compliant systems. Problems are detected late in the project, when the descriptors are integrated and executed, together with the components in the system.

Unvalidated descriptor files are expensive. Errors can arise at the most inconvenient times — such errors can lead to serious problems during integration, deployment, or run-time. Unvalidated descriptor files can also result in rejection during the certification process.

These errors are difficult and expensive to find with existing methods, which are inadequate for such undertakings. Methods that customers currently use include DTD validation and the SCA CF. Section 3 describes these methods. Section 4 classifies the errors and explains how the current methodologies perform in resolving the errors. Section 4 also qualifies the impact that the different errors can have on the progress in the development of an SCA-compliant system.

All these reasons increase the cost associated with SCA development: XML descriptor files are expensive to write, are error-prone without proper validation, and usually contain many errors that linger in the code until the product is integrated and tested — and afterwards.

Unvalidated descriptor files increase the risk associated with projects, which can derail or push projects significantly over budget.

3 EXISTING METHODS

3.1 DTD Validation

DTD Validation validates the syntax of the XML descriptor files. This can only catch a small percentage of the errors. It does not evaluate cross-references within or between files. Essentially, it verifies whether the descriptor files are well-written, not whether the contents of the files are semantically correct.

Moreover, the XML DTDs are intentionally incomplete with respect to the SCA semantics; that is, many SCA rules are not included in the DTDs because including them would make the DTDs too big.

3.2 SCA Core Framework

The SCA CF is the component management layer in an SCA application. This layer needs to interpret the XML descriptor files to find the information that it needs to deploy, configure, and start the application.

Since the CF is not focussed on validation, it does not validate all the information in the descriptor files. The CF focusses on execution. This means that an error in an descriptor file is typically found because something does not download, start, connect, or behave as expected. These types of errors are expensive to find and debug. Once the error is found, the root cause is not immediately evident. Multiple download-start-debug cycles using the CF are needed to find the root cause of the problem. Each cycle can last anywhere from 1 to 15 minutes, depending on the size of the application and platform.

4 ERRORS

Errors in the descriptor files can have many origins. They can vary from simple typing mistakes to serious architectural inconsistencies.

There are four major classes of errors:

- **Syntax** — errors against the DTD
- **Completeness** — availability of all referenced files and interfaces
- **Semantics** — availability of explicit rules as indicated in the standard
- **Deployment** — validation of application against a platform for deployability

The following subsections further elaborate the different levels of errors, provide examples and look at error detection and the possible impact of such errors.

4.1 Syntax

Introduction

Syntax errors are violations against the format of the XML in the descriptor files. These violations might be against the syntax of XML itself or against the DTD.

Syntax errors have several forms, including

- Unclosed quotation marks (“
- Missing XML tags
- Missing XML attributes
- Incorrect values for tags and attributes

Error Detection

XML errors can be detected by DTD validation tools. These tools highlight the errors by providing line numbers of the source of the error.

The SCA CF also performs DTD validation, usually during system startup.

Impact

DTD validation is relatively simple. Many customers, however, do not use it since it requires some effort to install and configure correctly.

Validation through the SCA CF is painful. Considerable setup time is required to start the CF with the appropriate files. The CF usually runs on an embedded target, which means that the XML descriptor files need to be moved to that target. A single modify-load-debug cycle can take up to 15 minutes. The CF's main goal is to start and control an embedded application, not validate XML. Hence, its error-reporting capabilities are limited and very expensive compared to using more appropriate tools for descriptor file validation.

4.2 Completeness

Introduction

Violations against completeness occur when the set of descriptors is missing files or when references are not completely defined. A second class of completeness violations relates to cross-references within one file or between files.

The Domain Profile contains a set of descriptor files for the Software Profile and a set of descriptor files for the Device Profile as mentioned in section 2.

The Software Profile starts at the SAD and then branches out to include the other file types to build a complete application (waveform).

The Device Profile starts at the DCD and then branches out to include the other file types to build a complete hardware platform.

All descriptor files that are mentioned in the Domain Profile — from the SAD and DCD down — need to be provided. If they are not, the application or platform is incomplete and a run-time error will occur during startup.

The descriptor files contain cross-references by unique identifiers (UUIDs). These cross-references can be within or between files. Cross-references can have up to three levels of indirection, which makes resolving them all the more complicated.

Examples of cross-reference problems include the following:

- The SAD contains a section where the files are listed for all the components and a section where instances are listed that reference back to the file listing.
- The implementation of a component can point to a specific implementation of another component (for example, a dynamic link library) as a dependency.
- The SAD specifies connections between ports on instances. These ports need to be defined in the SCD of the components that define the instances.
- The SAD can make connections to ports based on *deviceusedbythiscomponentref*. These type of connections point to a *componentinstantiation* with a *usesdevice* relationship. The *usesdevice* points to an allocation attribute on a device.
- A *componentinstantiation* in the SAD or DCD can override a property defined on the component or device definition.
- An SCD points to interfaces that are used on ports. These interfaces need to be defined in proper IDL definitions.

Error Detection

Completeness errors are not detected by DTD validation. An extra level of knowledge of the contents of the descriptor files is needed for this. Errors against completeness are usually found when starting the application or platform.

Missing cross-references are more difficult to resolve. Most CFs will indicate that a cross-reference is not satisfied; however, it's up to users to find the root cause of the missing target of the reference.

Impact

Missing files are typically easy to resolve: the CF simply states that a file is missing. Users can correct this relatively quickly, then reload the application or platform.

Missing cross-references are more difficult to resolve. Users need to find the root cause of the missing reference and resolve it. This requires detailed knowledge of the content of the descriptor files. Frequently, it is not straight forward. Users sometimes spend days resolving problems in this area. To complicate matters, cross-references frequently use unique identifiers specified with UUIDs, which are difficult to read and compare.

Errors in the XML can be resolved. However, these errors might be hiding a larger problem: an incorrect design. Component-based systems can be designed and built to be completely independent. However, if the individual components are under-specified, then problems can arise during integration — late in the development process. For example, the high-level design calls for a component with a specific interface, but the implementation has been built without this interface. These types of errors are expensive to resolve, especially when they are found during integration.

4.3 Semantics

Introduction

Semantic errors are an extension of Completeness, which is described in the previous section. Completeness ensures all the pieces of the puzzle are present. Semantics ensures all the pieces of the puzzle actually fit together.

The semantics of an SCA-based system concerns the relations and connections between components. Examples include

- connections between ports/interfaces have the correct interfaces, version, conjugation, and type
- correct interfaces on CF elements and assembly controller
- repository ids are consistent with interfaces

Error Detection

Detection of these types of errors can only be done at run-time. The CF starts the platform and the application. The different component instances are connected through the device managers or ApplicationFactory, depending on where the connections are defined.

If all is well, the connections work without a problem. However, if one side of a connection is implemented using the wrong interface, run-time exceptions can occur. These exceptions can occur during startup, but also at unexpected times during run-time of the system.

Impact

Like the Completeness type of errors, Semantic errors are only verified at integration time, which makes these types of errors expensive to resolve. Indeed, a re-design of a component might be necessary. This is an expensive change when the error is found late in the software development process.

Semantic errors are the most difficult to detect. Without proper tools they can only be detected when the system is executed on an embedded platform. The cost of this is not only a modify-load-debug cycle, but once a problem is found, resolution is not simple. It requires a change at one end of a connection. Moreover, it could require consultation with the system architecture to decide the correct course of action.

Semantic errors are potentially dangerous and should be avoided at all cost.

4.4 Deployment

Introduction

Deployment errors concern the dynamics of a system during configuration and deployment. The SCA defines applications independently from the platform. The SCA CF dynamically discovers the capabilities of the platform during deployment: it decides which software

component is downloaded and executed on a specific processor.

There are multiple aspects of deployment, including

- Dependencies concerning the operating system and processor requirements of a specific component implementation
- Capacity requirements from components to devices, for example, processing or throughput capacity
- Uses relationships where a component requires the use of a specific device
- Collocation requirements where two component instances are required to execute on the same device

Error Detection

Error detection for deployment issues happens during system startup. The CF dynamically explores the relationships between the application and the platform and tries to meet the requirements expressed therein.

Impact

These types of errors can only be caught during integration. This leaves the project open to expensive misinterpretations that possibly require software or even hardware modifications.

Deployment is not entirely deterministic. There is often more than one deployment possible given a certain platform and application.

It is difficult to enforce certain deployment schemes and hence it is difficult to verify that all required deployments are possible.

Deployment verification needs to be done manually. The system is started and the developer then evaluates whether all the instances are executed on the intended devices by inspecting the information provided by the CF.

5 ZELIGSOFT COMPONENT ENABLER

Zeligsoft Component Enabler is a visual modeling tool that provides the facilities for users to conveniently model, validate, and generate SCA-compliant systems. It provides a user-friendly, model-based workflow to define components and applications, devices and platforms.

A visual model is an important asset. The visual model describes the architecture, the starting point of development for an SCA compliant system. A proper architecture — proper consideration about the elements of the system and how they relate together — is critical. A proper architecture is the backbone of a software development process. It is needed to guide the team through development. It is the basis for code, documentation, discussions, and descriptor files.

The SCA is a standard with many rules that are not necessarily intuitive, especially to new users. A thorough understanding of the standard is needed to design SCA-compliant applications and platforms, and to be able to write the correct set of XML descriptors that specify them. Through its validation and generation feature, CE removes the need to be an XML expert. CE guides users through the pitfalls of the standard through the use of modeling. The tool helps users define the elements that comprise the system and highlights omissions or violations against the standard.

Once the model is created, it needs to be validated. As previously stated, the SCA standard defines many rules and regulations that the system needs to meet. Validation is an important aspect of modeling. It is comparable to syntax checking source code. A model with errors is like a piece of C code that does not correctly compile. It means that the system is not correctly described. Further work with the model can lead to undefined behavior, for example, during download and execution.

Furthermore, a model that does not correctly validate does not adhere to the standard. This means that the system might not pass certification test by JTEL, the JTRS certification authority.

6 XML IMPORT

CE provides visual modeling for SCA-compliant systems. One of the important features of CE is that it can create visual models based on existing XML descriptor files.

This feature builds up a complete semantic model of the system. Users can import single or multiple components or devices, an entire platform, an entire application, or an entire system consisting of multiple platforms and applications.

The import process starts a multi-step process where the user and the tool together validate the system described by the XML descriptors. CE manages the automated detailed tasks of syntactic validation, completeness check, and semantic validation — after which, users can visually inspect the model.

6.1 Syntactic Validation

The XML Import facility validates the XML for syntactic correctness. This involves validating the well-formedness of the XML, followed by validation of the XML against the DTD. This covers all of the errors as covered in section 4.1.

CE has a selective approach to syntactic validation: it ignores a syntax error if the error is not crucial in the understanding of content of the file. The result of this is that users are able to import descriptor files with certain syntax errors. These errors are highlighted as warnings during importation, and will continue to be highlighted when the model is validated against SCA compliance. Users are able to correct these errors through model-based workflows. An example of this type of error is an empty corbaversion tag.

Users are presented with an error message if a serious syntax error is encountered during importation, together with the line number that the error occurred on. Users need to correct the syntax error in the XML descriptor files before continuing. An example of this type of error is missing or invalid tags.

6.2 Completeness Check

The completeness check covers all of the errors as explained in section 4.2. It checks whether all the required files are included in the set of XML descriptors that is being imported.

Once file inclusion has been validated, it verifies whether all the cross-references between the different sections in the files are correct.

The completeness check flags missing files in the set of descriptors as a violation. It prompts users to provide the missing files. Missing IDL interfaces are also flagged.

Incorrect cross-references are indicated with an error message, which shows the type of error and its line number. Some of these errors need to be corrected in the XML; some of the errors can be corrected through model-based workflows.

An example of a model-based workflow follows: a SAD contains a connection that references a port on a component instance that does not exist in component definition in the SCD. The XML Import process highlights the missing port through an error message. Users can then create the port on the proper component and re-import the SAD.

6.3 Semantics Check

The semantics check covers the errors as discussed in section 4.3. A complete model of the content contained within the descriptor files is needed. The XML Import feature builds a complete model, based on the content of the descriptor files. Semantics validation is only performed after a complete model has been built.

The SCA Validation feature performs connection validation. It iterates through the entire model and evaluates the connections in the application (the SAD) and the node (the DCD). It also verifies whether the repository id of a component or device is provided by one of the ports or interfaces.

SCA Validation is performed after the XML Import process. Users are presented with a list of violations, together with suggested solutions. Each violation is hyperlinked to the location of the error. Users need to resolve these through model-based workflows.

6.4 Deployment Validation

All the previous types of validation discussed are concerned with static information. Deployment validation looks at the components and how they interact: it is concerned with how the application relates to the platform during run-time, which is dynamic information.

This dynamic information can be partially validated in a static fashion. Validation does not simulate the real deployment: it validates the relationships that will be used during deployment. This type of validation can never be 100% complete because it depends on capacity properties that can change dynamically. However, it does provide a valuable assessment of whether the application can be deployed in an ideal environment. If deployment validation fails, the application will never deploy on a real target.

The current version of CE does not perform Deployment Validation. This feature is scheduled for a future release.

6.5 Visual Validation

CE provides users with a graphical representation of the content in the descriptor files after the XML has been imported. Users can easily read and understand the visual representation. Inspecting the visual representation allows users to verify, for example, whether the ports that are connected in the descriptors are really supposed to be connected.

The visual representation also provides information about extra ports on components that are not connected, components that are not used, and so forth. The visual model ties all the information in all the descriptor files together into one convenient representation.

6.6 Zeligsoft Experience

The Zeligsoft team has gained much experience in helping customers with this validation process. We have helped customers who are developing systems for the JTRS clusters — as well as customers outside of the clusters — to validate their descriptor files.

While helping these customers, every set of descriptor files that we have examined so far has had a significant number of errors in them. The errors that we found range from missing corbaversion tags, to missing ports in SCD files, missing SCD files, wrong connections in the SAD, incorrect allocation attributes, and so forth.

Helping our customers has clearly indicated that descriptor file validation is a necessary task that will reduce the cost and risk of SCA system development and improve the quality of component-based software development within the context of the SCA.

7 SUMMARY

The information in the descriptor files is normative and mission critical for any SCA-compliant project. Incorrect information in the descriptor files is often an indication of larger problems in the architecture or implementation of the components. These problems need to be resolved as soon as possible. If these problems are not resolved, they will appear at unexpected points during the development process.

Frequently, these problems are only resolved during the integration process. At this stage of the software development process, resolution will consume excessive resources, as progress is blocked until the integration is successful.

Problems can be prevented and risk for the overall project can be reduced by validating the XML descriptor files. Validation prevents errors from spreading. It provides a solid understanding of the architecture of the software system.

Zeligsoft Component Enabler validates the syntax, completeness, and semantics of the XML descriptors for an SCA-compliant system. CE also provides a visual representation of the architecture. These features are important for any software project that builds SCA-compliant systems.

The XML Import process results in a visual model, which can assist users in improving their descriptor files. Once the model has been built, it can be used in the rest of the development process. Using a visual modeling tool increases the understanding of the entire development team and improves the efficiency and quality of the overall development process.

8 REFERENCES

- XML
<http://www.w3.org/XML/>
- SCA
http://jtrs.army.mil/sections/technicalinformation/fset_technical_sca.html
- JTRS
<http://jtrs.army.mil>
- JTEL
<https://jtel.spawar.navy.mil/main.asp>
- Zeligsoft
<http://www.zeligsoft.com>



Contact Information

Website: www.zeligsoft.com

Email: info@zeligsoft.com

Toll-free (North America): 1-800-ZELIGSW (1-800-935-4479)

Direct dial: +1 819-684-9639